INTELLEC® SERIES III MICROCOMPUTER DEVELOPMENT SYSTEM CONSOLE OPERATING INSTRUCTIONS

Manual Order Number: 121609-001 Rev. A

8/19/80

Intel Corporation, 3065 Bowers Avenue, Santa Clara, California 95051

This manual provides console operating instructions for the Intellec Series III Microcomputer Development System, an advanced system that provides dual operating environments, one for 8086/8088-based software and another for 8080/8085-based software.

This manual is designed to support new users as well as those who are already familiar with microcomputers.

Operation of this system requires version 4.1 or later of the ISIS-II operating system. Wherever 8086-based software is referenced in this manual, the information applies equally to 8088-based software.

This manual contains seven chapters and seven appendixes:

- Chapter 1, System Overview, describes the keyboard, control panel, screen, disk drives, and configurations supported.
- Chapter 2, System Operation, describes system start-up and shut-down, and disk formatting tutorials.
- Chapter 3, Device/File Management, describes device/file naming formats, device/file accessing, and disk directories.
- Chapter 4, Console Commands, describes and shows examples of the ISIS-II commands for storing, identifying, and manipulating your programs.
- Chapter 5, The Monitor, describes and shows examples of the Monitor commands for 8080/8085 program debugging.
- Chapter 6, DEBUG-86, describes and shows examples of the DEBUG-86 commands for 8086 program debugging.
- Chapter 7, Error Messages, gives a listing of error messages and recovery.
- Appendix A, Hexadecimal Paper Tape Format, describes the paper tape format used by the Monitor.
- Appendix B, Hexadecimal-Decimal Conversions, provides hexadecimal-to-decimal and decimal-to-hexadecimal conversions.
- Appendix C, ASCII Codes, shows ASCII codes, their meanings, and their values.
- Appendix D, Summary of ISIS-II Console Commands, provides a listing of all ISIS-II console commands and their syntax.
- Appendix E, Summary of Monitor Commands, provides a listing of all Monitor commands and their syntax.
- Appendix F, Summary of DEBUG-86 commands, provides a listing of all DEBUG-86 commands and their syntax.
- Appendix G, Summary of Error Messages, provides a listing of error messages.

The Intellec Series III Microcomputer Development System is part of the Microsystem 80 iAPX family of processors.

Microsystem 80 Nomenclature

Over the last several years, the increase in microcomputer system and software complexity has given birth to a new family of microprocessor products oriented towards solving these increasingly complex problems. This new generation of

microprocessors is both powerful and flexible and includes many processor enhancements, such as numeric floating point extensions, I/O processors, and operating system functionality in silicon.

As Intel's product line has grown and evolved, its microprocessor product numbering system has become inadequate to name VLSI solutions involving the above enhancements.

In order to accommodate these new VLSI systems, we've allowed the 8086 family name to evolve into a more comprehensive numbering scheme, while still including the basis of the previous 8086 nomenclature.

We've adopted the following prefixes to provide differentiation and consistency among our Microsystem 80 related product lines:

iAPX — Processor Series
iRMX — Operating Systems
iSBC — Single Board Computers
iSBX — MULTIMODULE Boards

Concentrating on the iAPX Series, two Processor Families are defined:

iAPX 86 — 8086 CPU based system
iAPX 88 — 8088 CPU based system

With additional suffix information, configuration options within each iAPX system can be identified, for example:

iAPX 86/10	CPU Alone (8086)
iAPX 86/11	CPU + IOP (8086 + 8089)
iAPX 88/20	CPU + Math Extension (8088 + 8087)
iAPX 88/21	CPU + Math Extension + IOP (8088 + 8087 + 8089)

This nomenclature is intended as an addition to rather than a replacement for, Intel's current part numbers. These new series level descriptions are used to describe the functional capabilities provided by specific configurations of the processors in the 8086 Family. The hardware used to implement each functional configuration is still described by referring to the parts involved (as is the case for the majority of the 8086 information described in this manual).

This improved nomenclature provides a more meaningful view of system capability and performance within the evolving Microsystem 80 architecture.

Related Publications

For more information on the Intellec Series III Microcomputer Development System, see the following manuals:

- *Intellec Series III Microcomputer Development System Product Overview*, 121575
- *Intellec Series III Microcomputer Development System Programmer's Reference Manual*, 121618
- *Intellec Series III Microcomputer Development System Installation and Checkout Manual*, 121612
- *Intellec Series III Microcomputer Development System Schematic Drawings*, 121642
- *iAPX 86, 88 Family Utilities User's Guide for 8086-Based Development Systems*, 121616

- *ISIS-II CREDIT CRT-Based Text Editor User's Guide*, 9800902
- *The 8086 Family User's Manual*, 9800722
- *8086/8087/8088 Macro Assembly Language Reference Manual for 8086-Based Development Systems*, 121627
- *8086/8087/8088 Macro Assembler Operating Instructions for 8086-Based Development Systems*, 121628

Auxiliary Product Manuals

The following manuals describe auxiliary products:

- *PL/M-86 User's Guide for 8086-Based Development Systems*, 121636
- *Pascal-86 User's Guide*, 121539
- *MCS-80/85 Utilities User's Guide for 8080/8085-Based Development Systems*, 121617
- *8089 Macro Assembler User's Guide*, 9800938
- *ICE-86 In-Circuit Emulator Operating Instructions for ISIS-II Users*, 9800714
- *ICE-88 In-Circuit Emulator Operating Instructions for ISIS-II Users*, 9800949
- *iSBC 957A Intellec-iSBC 86/12A Interface and Execution Package*, 9800743
- *RBF-89 Real-Time Breakpoint Facility Operating Instructions for ICE-86 In-Circuit Emulator Users*, 9801018
- *Model 740 Hard Disk Subsystem Operation and Checkout*, 9800943

Notational Conventions

UPPERCASE	Characters shown in uppercase must be entered in the order shown. You may enter the characters in uppercase or lowercase.
italics	Italics indicate variable information, such as *filename* or *address*.
[ ]	Brackets indicate optional arguments or parameters.
{ }	One and only one of the enclosed entries must be selected unless the field is also surrounded by brackets, in which case it is optional.
{ } ...	At least one of the enclosed items must be selected unless the field is also surrounded by brackets, in which case it is optional. The items may be used in any order unless otherwise noted.
...	Ellipses indicate that the preceding argument or parameter may be repeated.
punctuation	Punctuation other than ellipses, braces and brackets must be entered as shown. For example, the punctuation shown in the following command must be entered: `SUBMIT PLM86(PROGA,SRC,'9 SEPT 81')`
input lines	In interactive examples, input lines and user responses are printed in white on black to differentiate input lines from sysytem output.

CONTENTS (Cont'd.)

ILLUSTRATIONS

TABLES

CHAPTER 1
SYSTEM OVERVIEW

The Intellec Series III is an advanced microcomputer development system that provides two execution environments and two development facilities: an 8086 execution mode and applications debugger, and an 8080/8085 execution mode and debugger.

This chapter provides an overview of:

- The basic system
- System software components
- Dual execution modes
- The Intellec terminal
- Disk drives
- Disk drive configurations
- Disk characteristics
- Disk files

The Basic System

The basic system, shown in figure 1-1, contains the following:

- An 8085 processor
- 64k bytes of memory available to the 8085 processor
- An 8086 processor
- 128k bytes of memory available to the 8086 processor
- 8080/8085 I/O interfaces for teletypewriter, paper tape punch and reader, line printer, Universal PROM programmer
- An integral single-density flexible disk drive

Figure 1-2 shows a system with an external dual drive flexible disk unit; figure 1-3 shows a system with a hard disk subsystem.

System Software Components

The following system software components are supplied with the Intellec Series III Microcomputer Development System:

- Monitor—provides system start-up, peripheral device I/O, and debugging of your 8080/8085-based software as described in Chapter 5.
- ISIS-II—provides disk I/O, file management, and program execution in the 8080/8085 or 8086 execution mode as described in Chapters 3 and 4.
- RUN 8086—provides 8086 execution mode as described in Chapter 4.
- DEBUG-86—Provides symbolic debugging of your 8086-based software as described in Chapter 6.

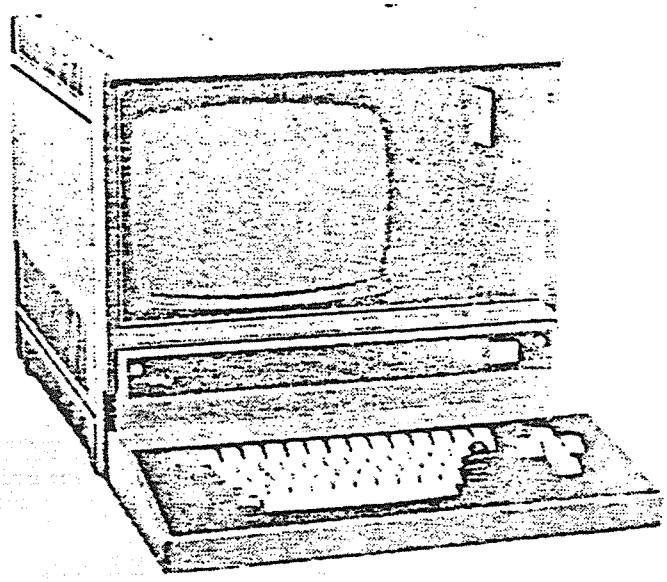


Figure 1-1. Basic System 121609-14

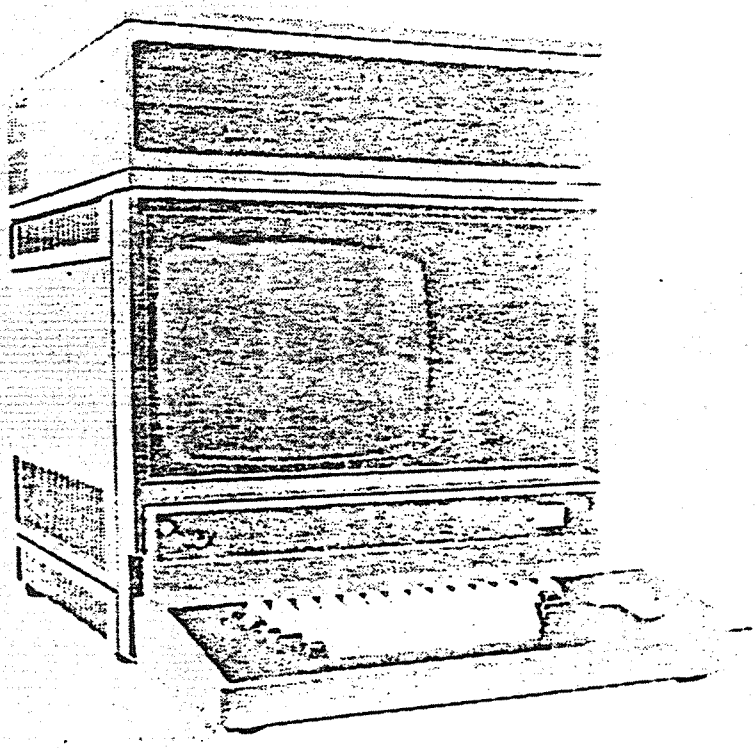


Figure 1-2. System With External Flexible Disk Drive Unit 121609-1

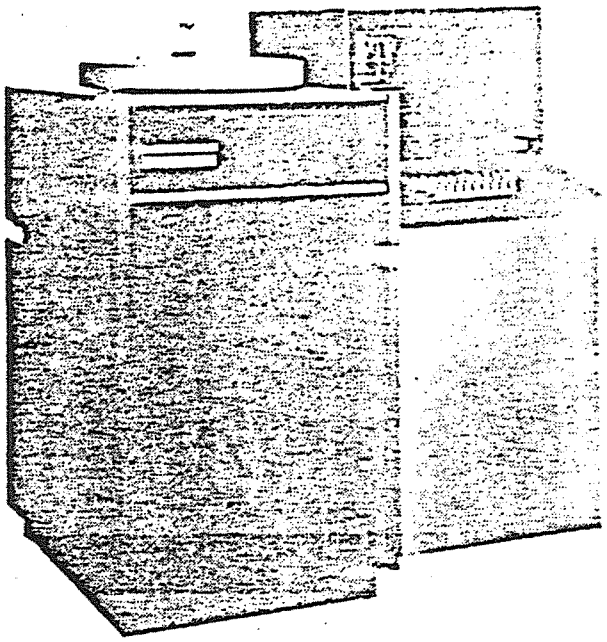


Figure 1-3. System With Hard Disk Subsystem 121609-2

Dual Execution Modes

8080/8085 Execution Mode

The following functions operate in the 8080/8085 execution mode:

- All ISIS-II console commands except RUN and its subcommands
- Monitor commands

You load and execute your 8080/8085 executable programs by simply entering the name of your program.

8086 Execution Mode

The following functions operate in the 8086 execution mode:

- The RUN command, which invokes the 8086 mode
- The RUN subcommands, DATE, WORK, and EXIT
- DEBUG-86 commands

You load and execute your 8086 executable programs by entering the RUN command and the name of your program.

To use ISIS-II console commands, you must first exit to the 8080/8085 mode described under the RUN and EXIT commands at the end of Chapter 4.

Each program displays a prompt character, as follows:

Program	Prompt	Mode
ISIS-II	- (hyphen)	8080/8085
Monitor	. (period)	8080/8085
RUN	> (angle bracket)	8086
DEBUG-86	* (asterisk)	8086

The Intellec Terminal

This section describes the display area, keyboard, and control panel of the Intellec terminal.

The Display Area

The video screen, shown in figure 1-4, has a display area that is 80 characters wide and 25 lines long.

Every character typed at the keyboard is displayed on the screen. A cursor (a blinking underscore) indicates where the next character will be entered.

You may use special features of the display when creating files with the CREDIT Text Editor (see the *ISIS-II CREDIT CRT-Based Text Editor User's Guide).*

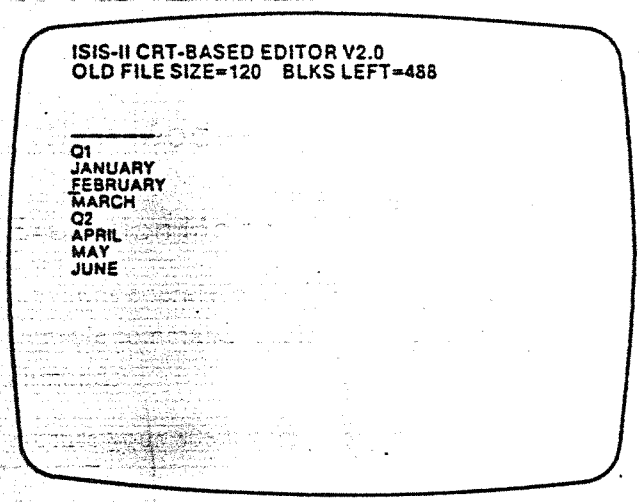


Figure 1-4. The Display 121609-3

The Keyboard

The keyboard, shown in figure 1-5, is your interface with the system. From the keyboard you control the system, enter data and commands, and request data.

The data you enter at the keyboard is stored in a line editing buffer until you press the RETURN key or enter 122 characters. You can edit the contents of the line editing buffer with the line editing characters described in Chapter 3, Line Editing.

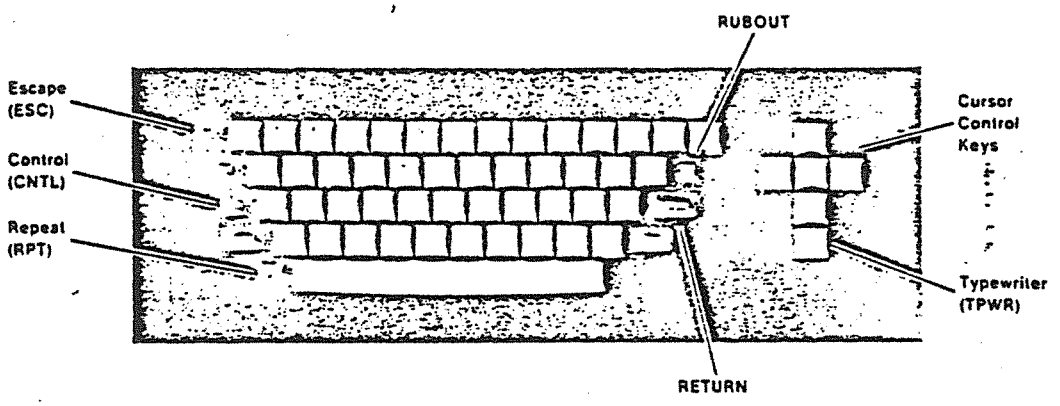


Figure 1-5. The Keyboard 121609-4

The keyboard is a typewriter-style electronic keyboard. In addition to the standard typewriter keys, the keyboard has other keys that perform specialized functions, as follows:

CNTL	The CNTL (Control) key changes the function of certain predefined alphabetic keys. A key whose function is changed by the CNTL key is called a control character. Control characters are defined throughout the manual. Examples are CNTL-R, CNTL-S, and CNTL-Q. To enter a control character, hold down the CNTL key and type the designated letter. A control character is entered into memory as one character, but is displayed with an up arrow, as in ↑R.
ESC	The ESC (Escape) key terminates the line edited input in 8080/8085 mode. The ESC key displays as a dollar sign ($).
RPT	The RPT (Repeat) key provides multiple entry of other keys. When RPT and a second key are held down, the function of the second key is repeated until the RPT key is released. For example, to delete several characters from the current input line, press both the RPT and the RUBOUT keys until the desired number of characters are deleted. The RPT key functions with all keys except the CNTL, SHIFT, HOME, and TPWR keys.
RUB OUT	The RUBOUT key deletes the preceding character from both the display and the line editing buffer. Repeated usage is allowed.
RETURN	The RETURN key enters the carriage return and line feed characters. The use of the RETURN key in examples of input lines is indicated by <cr>.
TPWR	The TPWR (Typewriter) key provides lowercase entry (latched position) or uppercase entry (unlatched position) of alphabetic characters. The TPWR key functions with the alphabetic keys only.
HOME	The HOME key is used with the CREDIT Text Editor (see the *ISIS-II CREDIT CRT-Based Text Editor User's Guide).*
(arrows)	The four keys with arrows are cursor control keys and are used with the CREDIT Text Editor.

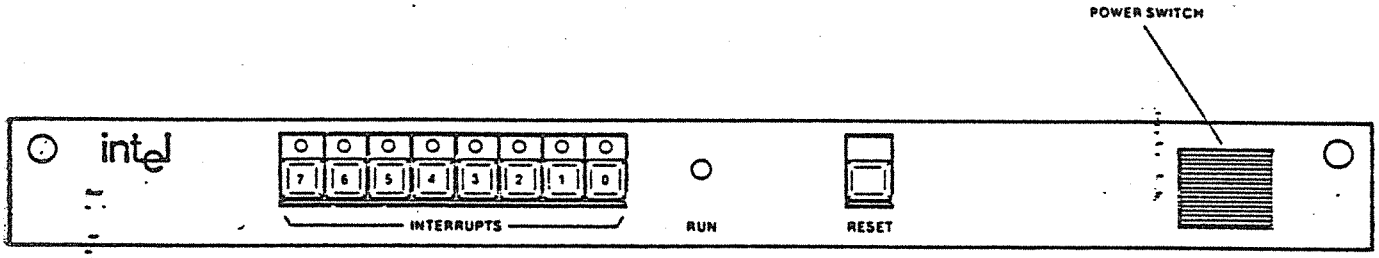


Figure 1-6. The Control Panel 121609-5

The Control Panel

The system control panel, shown in figure 1-6, contains the following switches and/or indicators (right to left):

POWER — The POWER switch (square white button) turns on the power to the basic system console including the integral disk drive and gives control of the system to the Monitor.

RESET — If a system disk is in drive 0, the RESET switch loads the ISIS-II operating system into 8080/8085 memory and gives control of the system to ISIS-II; if a system disk is not in drive 0, it restarts the Monitor.

RUN — The RUN indicator light remains on while system power is on and the 8080/8085 processor is not in a halt state.

INTERRUPTS

0 — Interrupt 0 is a manual interrupt. When you press interrupt 0, processing terminates and control of the system is transferred to the Monitor.

In 8086 mode, you can use CNTL-D to interrupt processing and enter DEBUG-86.

1 — Interrupt 1 is a manual interrupt. When you press interrupt 1, processing terminates and control of the system is transferred to ISIS-II if a system disk is in drive 0.

In 8086 mode, you can use CNTL-C to interrupt processing and return control to RUN or ISIS-II depending on how RUN was invoked.

2 — Interrupt 2 is reserved for system usage.

3,4,6,7 — Interrupts 3, 4, 6, and 7 can be used to generate Multibus interrupts if you make allowances for software products contained in your system.

5 — In 8086 mode, interrupt 5 must not be used.

Disk Drives

Hard disk subsystem controls are described in Chapter 2 under Operation of Systems Containing Hard Disk Drives.

Flexible disk drives are shown in figure 1-7. The front of each disk drive consists of:

- A disk drive door
- A drive door release button, which opens the drive door and releases the disk for removal
- A drive indicator light, which is lit during disk input/output operations.

In addition, the external drive units shown in figure 1-7 (b) and (c) contain a power switch and on/off indicator light

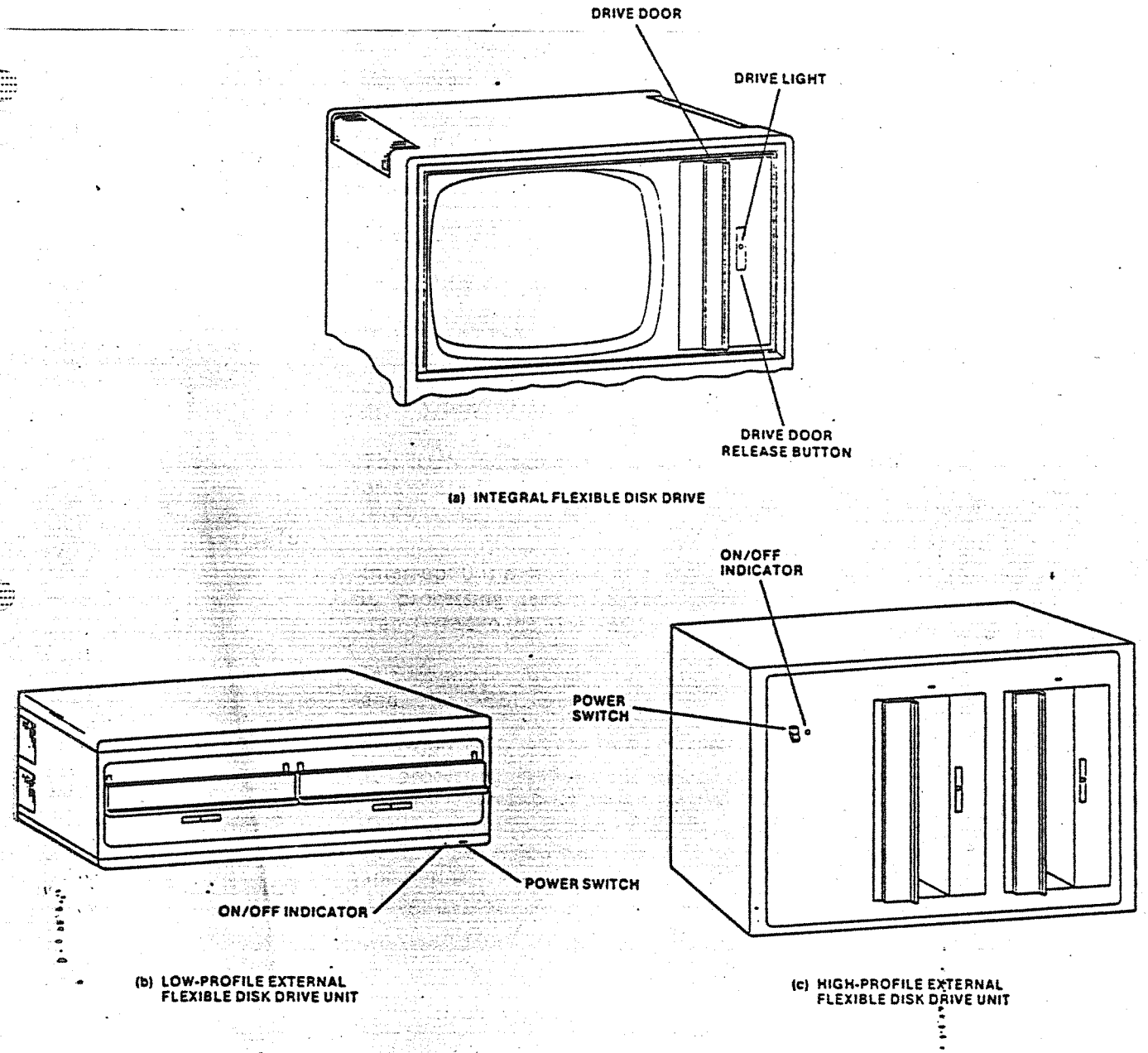


Figure 1-7. Flexible Disk Drives

121609-15

Disk Drive Configurations

The Series III supports up to eight drives, including single- and double-density flexible disk drives and hard disk drives. Table 1-1 describes the possible disk drive configurations and drive numbers for each.

You will note in table 1-1 that hard disk drives are always numbered drive 0 for the fixed hard disk platter, and drive 1 for the removable hard disk platter. Flexible disk drives, however, are assigned drive numbers according to the system configuration.

Table 1-1. Series III Disk Drive Configurations

Configuration	Drive Numbers									
	0	1	2	3	4	5	6	7	8	9
H + D	H-F	H-R	•	•	D	D	(D)	(D)		
H + IS	H-F	H-R	•	•	IS	•				
H + D + IS	H-F	H-R	•	•	D	D	(D)	(D)	IS	•
H + ID	H-F	H-R	•	•	ID	•			IS	•
H + ID + D	H-F	H-R	•	•	ID	•				
D	D	D	(D)	(D)			D	D		
IS	IS	•								
D + IS	D	D	(D)	(D)	IS	•				
ID	ID	•								
ID + D	ID	•	D	D						

H = Hard disk
F = fixed platter of hard disk
R = removable platter of hard disk
D = Double-density flexible disk
IS = Integrated single-density flexible disk
ID = Integrated double-density flexible disk
• = Not available
Parentheses ( ) indicate optional drives within the particular configuration.

Disk Format Characteristics

Flexible Disks

The flexible disk used with the Intellec Series III Microcomputer Development System contains 77 tracks each. A flexible disk formatted in a single-density disk drive has 26 sectors per track; a flexible disk formatted in a double-density disk drive has 52 sectors per track. Each sector contains 128 bytes.

Once a flexible disk is formatted in a single- or double-density disk drive, it must be reformatted to be used in a drive of a different density. A flexible disk is shown in figure 1-8.

Hard Disks

Hard disk platters contain 400 tracks on each of two surfaces. Each track has 36 sectors of 128 bytes. The system translates this data format into 200 logical tracks, with 144 logical sectors per logical track. Each hard disk subsystem contains two hard disk platters, a fixed platter in drive 0 and a removable cartridge in drive 1. A removable cartridge is shown in figure 1-9.

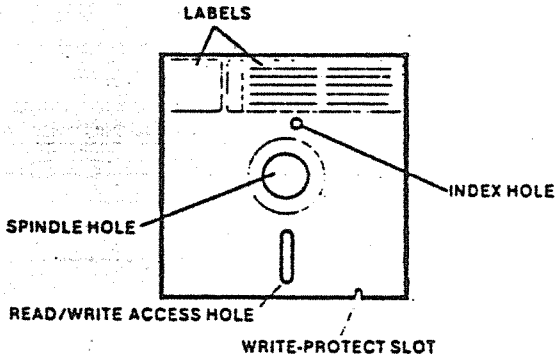


Figure 1-8. Flexible Disk 121609-7

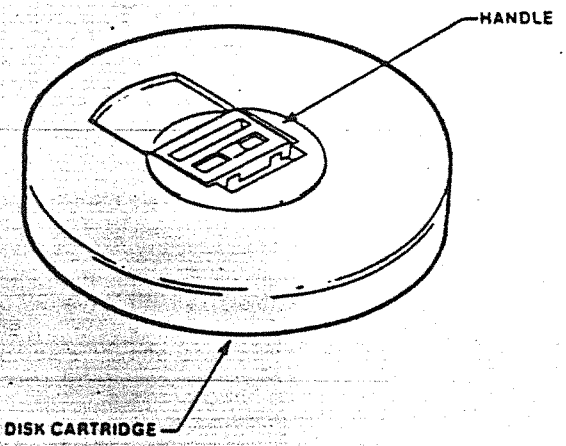


Figure 1-9. Hard Disk Cartridge 121609-8

Types of Disk Files

A disk is either a system or non-system disk depending on the ISIS-II files contained on it:

- A system disk contains at least the files necessary to boot the operating system.
- A non-system disk contains only the files necessary for the creation and storage of files, leaving more space for data than on a system disk.

When the system is reset with an ISIS-II system disk in the appropriate drive, the operating system initializes and takes control of the system.

At system start-up, only the essential ISIS-II files are loaded into memory. ISIS-II command files remain on disk until you enter a command that calls them. The required programs are then loaded into memory and executed. After the command program has completed its functions, the memory it was using is again available. This technique gives you the full capabilities of the operating system and lets you reserve most of the memory space for your work.

Each file on a disk has a name. ISIS-II program files come with assigned names; you name each file you create. To access a file, you need only specify its name, not its address.

The four basic types of files are:

- Format files that are used by ISIS-II for disk formatting.
- System files that contain both the basic system programs and the command programs
- Program files that you create
- Data files that are used by your programs or by ISIS-II

CHAPTER 2
SYSTEM OPERATION

This chapter provides tutorial information on how to operate the system, including the following functions:

- Care of disks
- Operating precautions
- Flexible disk insertion and removal
- Start-up and shut-down procedures for systems containing flexible disk drives
- Flexible disk formatting procedures
- Cold start and hard disk formatting procedures for systems containing a hard disk unit
- Hard disk cartridge installation and removal
- Subsequent start-up and power-down procedures for systems containing a hard disk unit

The first half of the chapter describes systems containing flexible disk drives; the second half describes systems containing hard disk drives.

Operation of Systems Containing Flexible Disks

Care of Flexible Disks

The flexible disk is a cost-effective and convenient medium for the storage of data. With proper care, you can ensure continued trouble-free reading and writing of flexible disk files. Specific precautions are:

- Return the flexible disk to its envelope when not in use
- Do not touch or clean the recording surface
- Do not smoke when handling the flexible disk
- Do not bend the flexible disk or use paper clips or other mechanical devices on it
- Use a felt tip pen on the disk label, not pencil or ball point pen

For information concerning the operating and storage environment, see the *Intellec Series III Microcomputer Development System Installation and Checkout Manual*.

Flexible Disk System Operating Precautions

The following actions can damage or modify the contents of a flexible disk:

- Turning on or turning off the power to the system or to an external disk drive unit with a flexible disk already inserted in the drive.
- Opening the disk drive door while the indicator light on the drive door release button is on.
- Removing a disk while the system is not at the ISIS-II level (when the ISIS-II prompt character is not displayed).
- Pressing the RESET switch while writing information on the flexible disk (i.e., the indicator light on the drive door is on).

Flexible Disk System Start-up Procedure

To start up a system containing flexible disk drives only, follow these steps:

1. Turn on the power switch on the console control panel.
2. Press the RESET button. The system displays the Monitor sign-on message and prompt character (a period):

```
SERIES II MONITOR, Vx.y
.
```

   (x.y is the version and release number of the Monitor.)
3. If you have an external disk drive unit attached to your system, turn on the power to the drive unit.
4. Insert the ISIS-II system flexible disk in drive 0 as described in the next section, Flexible Disk Insertion.
5. Press the RESET button. This loads the ISIS-II operating system files from disk into memory.

The system displays the ISIS-II sign-on message and prompt character (a hyphen):

```
ISIS-II, Vx.y
-
```

(x.y is the version and release number of ISIS-II.)

6. The system is now ready to accept a command from the console.

NOTE

After you press RESET in step 5, the ISIS-II prompt (a hyphen) should be displayed. If the prompt displayed is a period (indicating that the Monitor is still in control), check for one of the following conditions: a non-system disk in drive 0, an incorrectly installed disk, or a disconnected drive.

Flexible Disk Insertion

Before you insert a flexible disk, be sure that power to the system (and to the external flexible disk drive unit if any) is turned on.

If the drive door has a vertical opening, insert the flexible disk with the read/write slot first and the write-protect tab down (see figure 2-1).

If the drive door has a horizontal opening, insert the flexible disk with the read/write slot first and the write-protect tab to the left (see figure 2-2).

When the disk is inserted, close the drive door.

Flexible Disk Removal

To remove a flexible disk from a disk drive, follow these steps:

1. Check that the last character displayed is the ISIS-II prompt character (a hyphen) indicating that ISIS-II is in control of the system.

   If the last character displayed is not a hyphen, press interrupt 1.
2. Check that the indicator light on the drive door release button is off (see figure 2-3).

   If the light remains on for more than 10 seconds and a read operation is not in progress, disengage the drive by pressing the RESET button, holding RESET pressed if necessary until the light goes out.

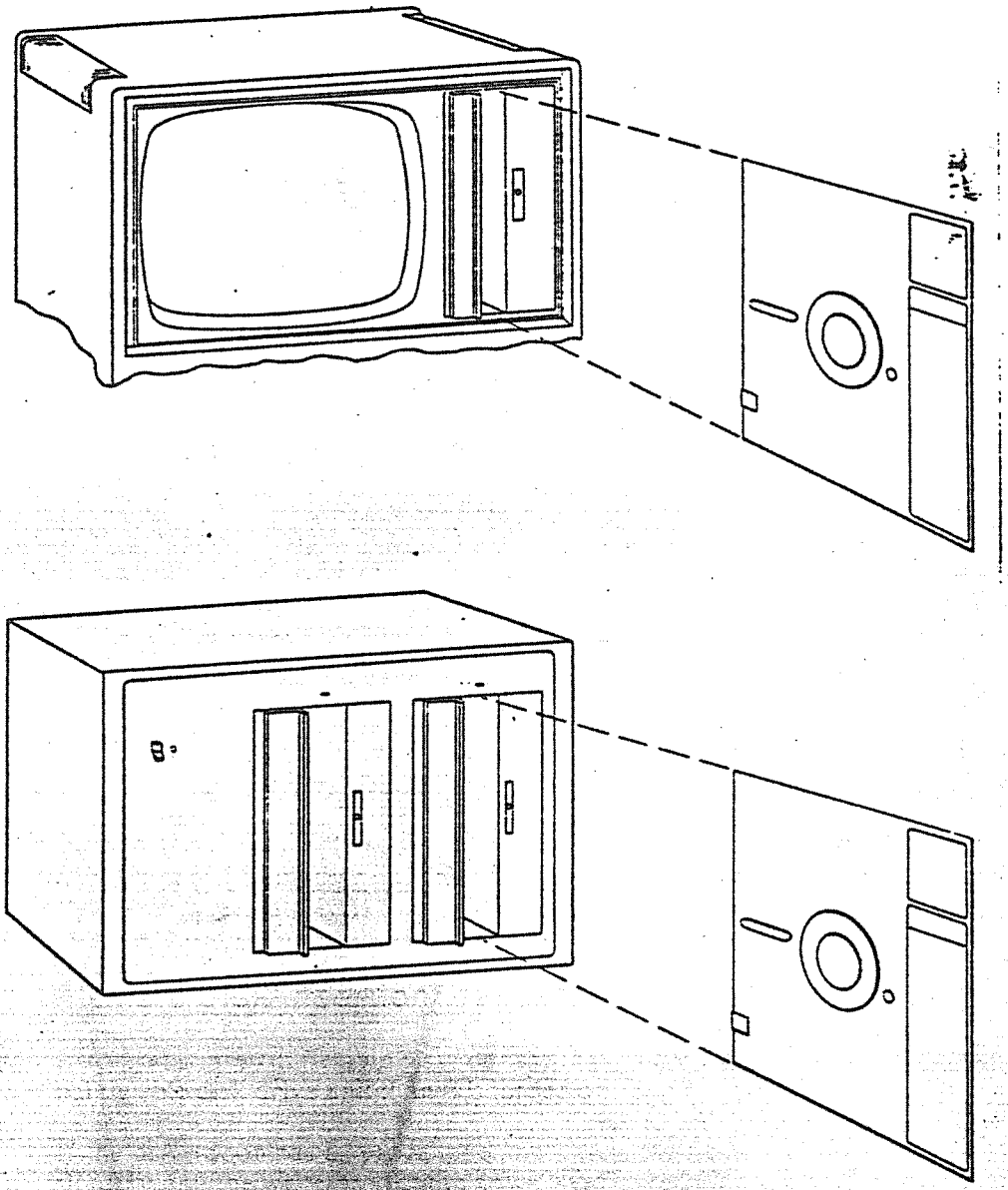


Figure 2-1. Flexible Disk Insertion, Vertical Position 121609-9

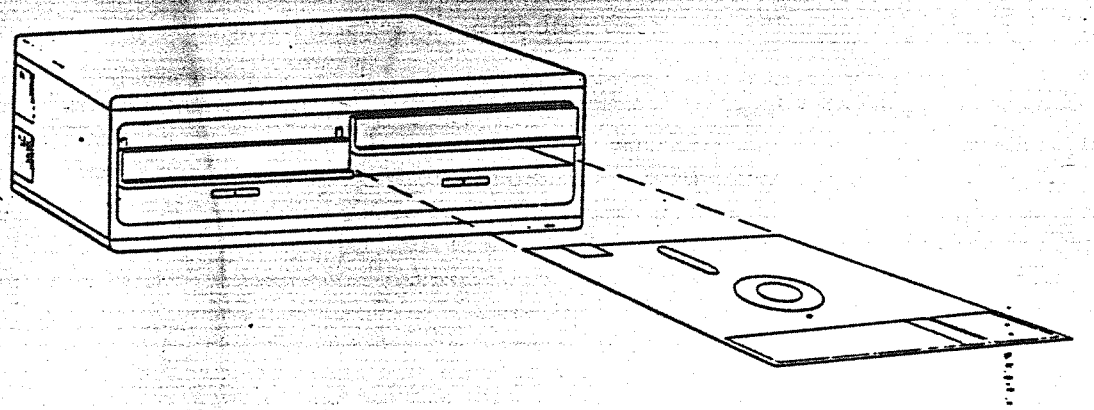


Figure 2-2. Flexible Disk Insertion, Horizontal Position 121609-10

3. Press the drive door release button. The door automatically opens and releases the flexible disk.
4. Remove the flexible disk and place it in its protective envelope.

Flexible Disk System Shut-Down Procedure

When you are ready to turn off the system, follow these steps:

1. Remove all flexible disks as described in the preceding section.
2. Turn off the power switch on the external disk drive unit if any.
3. Turn off the power switch on the console control panel.

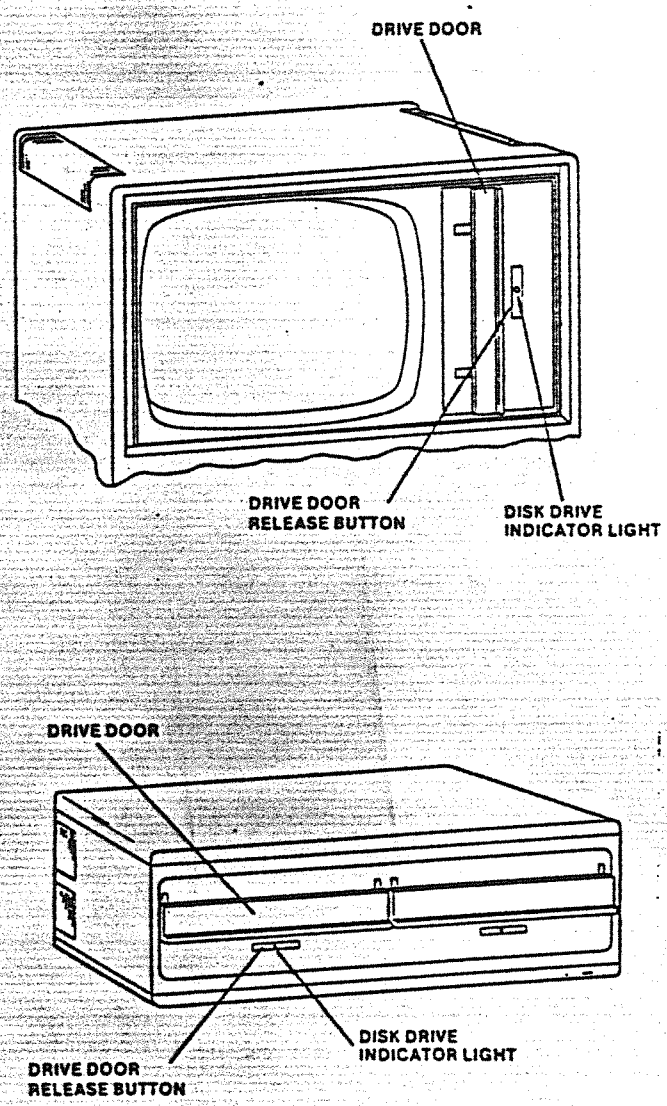


Figure 2-3. Drive Door Release Button 121609-12

Flexible Disk Formatting Procedures

Before using a blank disk, you must format it. Examples below provide step-by-step disk formatting instructions for systems with multiple flexible disk drives and for systems with a single flexible disk drive.

The flexible disks required are:

- A source disk, referred to as the system or source disk, that contains ISIS-II system files.
- A blank disk, referred to as the output disk.

  Before formatting a flexible disk, a reflective tab (provided with Intel disks) must be placed over an open write-protect slot, as shown in figure 2-4.

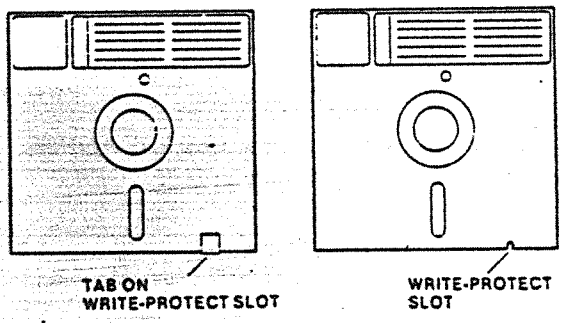


Figure 2-4. Flexible Disk Write-Protect Tab 121609-13

With the instructions given in example 1 you can produce any of three types of disks:

1. A basic system disk.
2. A basic non-system disk.
3. A duplicate back-up disk that contains all the files on the source disk. (It is a good idea to make a back-up copy of the important files on your disks.)

With the instructions given in example 2, you can produce types 1 and 3.

Example 1—Formatting a Flexible Disk in a System Containing Multiple Flexible Disk Drives

1. Apply power to the system. Press the RESET button. The system displays the Monitor sign-on message and prompt (a period).
2. Apply power to the external disk drive unit.
3. Insert the system disk into drive 0.
4. Insert the output (blank) disk into drive 1.
5. Press the RESET button.

   The system displays the ISIS-II sign-on message and prompt (a hyphen).
6. Enter one of the following commands:

   To format a basic system disk:

   -FORMAT :F1:MYDISK S<cr>

   To format and create a duplicate back-up disk:

   -FORMAT :F1:MYDISK A<cr>

   To format a nonsystem disk:

   -FORMAT :F1:MYDISK<cr>

7. The system displays the name of each file copied.
8. The disk is now ready to be used on the system.

For additional information and examples of the FORMAT command, see Chapter 4.

Example 2—Formatting a Flexible Disk in a System Containing a Single Flexible Disk Drive

It is recommended that all disks used on a single disk system be formatted as a system disk and that you write-protect your source system disk in case the two disks are switched.

1. Turn on the system power and press the RESET switch. The system displays the Monitor sign-on message and prompt character (a period).
2. Insert a system disk into the disk drive.
3. Press the RESET button. The system displays the ISIS-II sign-on message and prompt character (a hyphen).
4. Enter the following command:

   -IDISK :F0: MYDISK $S <CR>
5. The system displays:

   SYSTEM DISK
6. The system then displays:

   LOAD OUTPUT DISK, THEN TYPE (CR)

   The system waits for you to remove the system disk, insert the ouput disk, and press the RETURN key.
7. The system then displays:

   LOAD SYSTEM DISK, THEN TYPE (CR)

   The system waits for you to remove the output disk, insert the system disk, and press the RETURN key.
8. When the new disk is formatted it can be used on the system as a system disk. To produce a duplicate back-up disk, continue with the remaining steps in this procedure.
9. When the ISIS-II prompt character (a hyphen) is displayed, enter the following command:

   -COPY :F0:*.* TO :F0: <CR>

   where *.* is the wild card designation that matches all filenames. All files remaining on the source disk are copied to the output disk.
10. The system then displays:

    LOAD SOURCE DISK, THEN TYPE (CR)

    If the source disk is already inserted in the drive, press the RETURN key without swapping disks. If not, swap disks and press the RETURN key.
11. The system then displays:

    LOAD OUTPUT DISK, THEN TYPE (CR)

    Swap disks and press the RETURN key.
12. Repeat steps 10 and 11 in accordance with the messages displayed, if any.
13. The system displays the names of each file copied.
14. When the copy is completed, the new disk is a duplicate back-up disk.

For additional information on the IDISK and COPY commands, see Chapter 4.

Operation of Systems Containing Hard Disk Drives

Hard Disk Subsystem Controls

The front panel of the hard disk drive includes four backlighted operating switches and two status indicators (see figure 2-5 and table 2-1). A brush indicator and two cartridge holddown arms are mounted on the top of the disk drive. Two circuit breakers are positioned on the back panel.

Table 2-1. Hard Disk Drive Controls and Indicators

Control or Indicator	Function
MAIN Circuit Breaker (CB1)	Applies main ac to disk drive.
+34 VOLT Circuit Breaker (CB2)	Applies dc voltage to disk drive electronics; not accessible to operator (covered by switch plate).
START/STOP Switch/Indicator	Alternate-action switch with indicator. When pressed, applies power to spindle motor and initiates the first seek mode; indicator lights to indicate power is applied to spindle motor and spindle is rotating. When pressed the second time, removes power to spindle motor; indicator remains lighted until spindle stops rotating. NOTE The first seek mode is completely automatic and requires approximately 65 seconds to complete. In the event of a fault during this time, heads will automatically go to emergency retract and spindle will stop.
READY Indicator	Lights when spindle is up to speed, heads are loaded, and disk drive is ready for use.
ACTIVE Indicator	Lights when disk drive is actively engaged in any mode: direct seek (forward or reverse), return-to-zero seek, or read/write/erase.
FAULT Switch/Indicator	Lights when any fault (except power failure) exists. Pressing switch resets fault logic.
WRITE PROTECT/CART Switch/Indicator	Alternate-action switch with indicator. Prohibits writing or erasing on cartridge disk. Indicator lights to indicate that cartridge is protected.
WRITE PROTECT/FIXED Switch/Indicator	Alternate-action switch with indicator. Prohibits writing or erasing on fixed disk. Indicator lights to indicate that fixed disk is protected.
Brush Indicator	Indicates position of brush motor. Allows brush to be manually moved.
Cartridge Holddown Arms	Hold disk cartridge in place. Interlock circuits prevent arms from being lifted as long as spindle is rotating.

Care of Hard Disks

The hard disk drive assembly is extremely sensitive to contaminants on the hard disk platter surface. The head does not make contact with the disk platter, but rides about 1.14 microns above it. If contaminants such as a human hair (100 microns in diameter), a smoke particle (6.35 microns), fingerprints, or dust come between the disk drive head and the platter surface, the contact will usually destroy both the head and the disk.

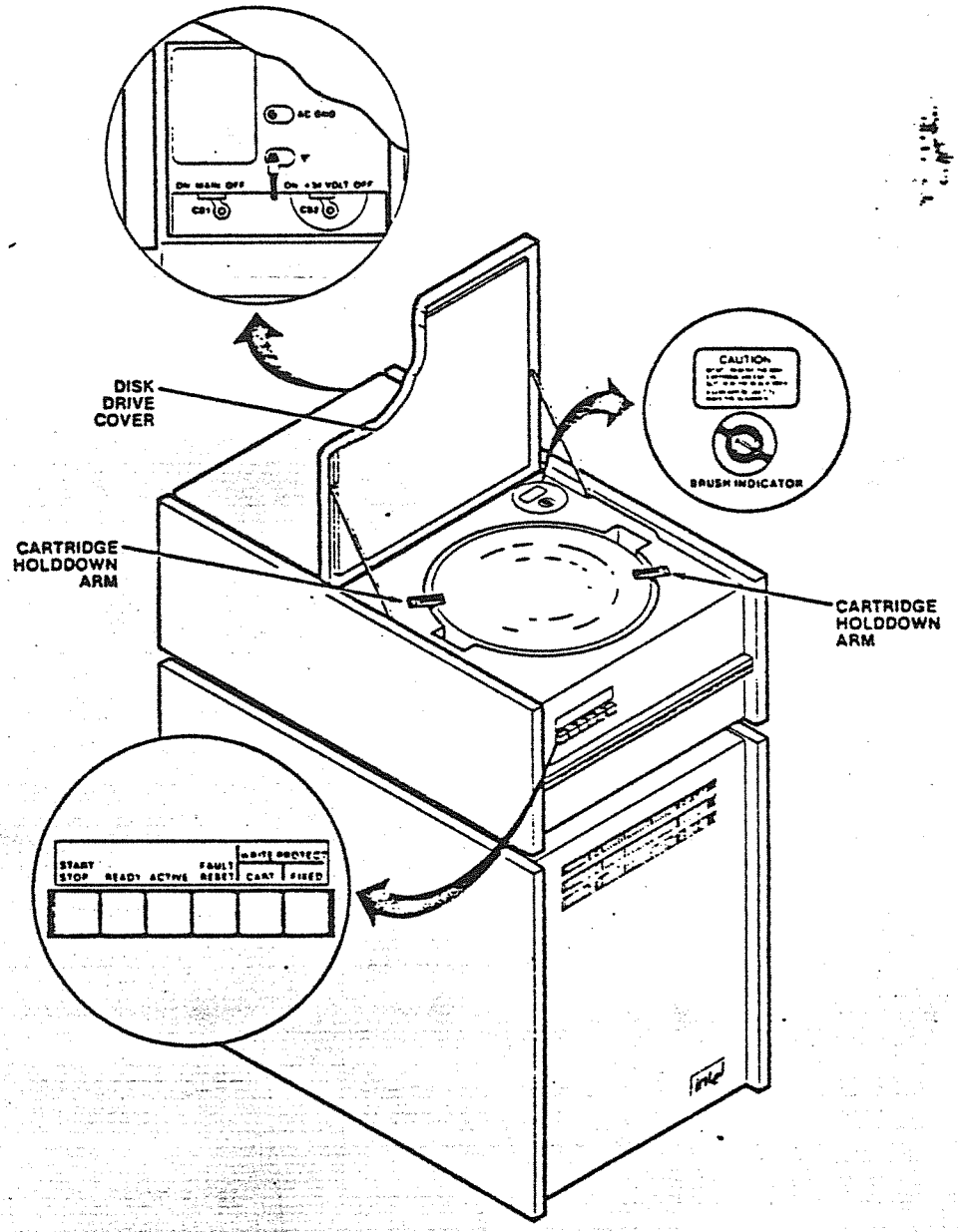


Figure 2-5. Hard Disk Drive Subsystem 121609-6

Follow these precautions to ensure proper operation and maintain data integrity:

- Clean any dust or dirt from the cartridge cover and drive chassis with a lint-free cloth.
- Allow nothing to touch the disk surface.
- Inspect the disk surfaces periodically with a bright, directional light. If the disk is dirty or scratched, it must be serviced by your Intel service representative.
- Keep liquids away from the hard disk drive.
- Do not smoke in the hard disk drive area.
- When you first bring a cartridge into a new operating environment, allow at least one hour for the cartridge temperature to stabilize.

Hard Disk Subsystem Operating Precautions

Follow these precautions to maximize subsystem performance;

- Always keep a cartridge installed and disk drive cover closed. This prevents dust and foreign particles from contaminating the disk cartridge or fixed disk mechanisms.
- If you hear a pinging or scratching sound (caused by head-to-disk contact), stop the disk drive by pressing the START/STOP switch and call a service representative.
- Always follow the disk cartridge installation and removal procedures described in this chapter.
- Never override any interlock switch or mechanism in the subsystem.

Hard Disk System Cold Start

Use this procedure when you first set up the hard disk subsystem for operation or when you need to re-format the system hard disk. For day-to-day operation, follow the procedure in the Hard Disk System Subsequent Start-up section.

Drive 0 contains the fixed hard disk platter, drive 1 contains the removable hard disk cartridge, and drive 4 is the integral flexible disk drive. (If your system has a single-density integral drive and an external flexible disk drive unit, drive 4 is the righthand drive on the external unit.

The system flexible disk and the fixed hard disk must contain the same version of ISIS-II. For Series III, you must use version 4.1 or later of ISIS-II. If the fixed hard disk does not contain the correct version, copy any files to be saved to another disk, then reformat the fixed hard disk.

Booting the system on is as follows:

1. Apply power to the system console and press the RESET switch.
2. Apply power to the external flexible drive unit if any.
3. Power up the hard disk subsystem as follows:
   a. Set the MAIN circuit breaker (CB1) to ON. Make sure the blower motors are running. If the blower motors do not turn on, call your service representative.
   b. Install a disk cartridge as described in the next section, Hard Disk Cartridge Installation.
   c. If the FAULT indicator comes on, perform steps 1 through 3 of the FAULT operation procedure in this chapter.
   d. Press the START/STOP switch to turn the spindle motor off.
4. Set the WRITE PROTECT switches to OFF unless write protection is desired (see figure 2-5).
5. Insert a system flexible disk in drive 4.
6. Press the RESET switch on the console.
7. The system displays the ISIS-II sign-on message and prompt character (a hyphen):

```
ISIS-II, Vx.y
-
```

   (x.y is the version and release number of ISIS-II.)
8. Press the START/STOP switch to turn the spindle motor on.

9. Wait until the READY indicator is lit (65 seconds).
10. At the console, enter the following command:

    -FORMAT :F0:ISIS.7 IS FROM x.y<CR>

    where x.y corresponds to the ISIS-II version and release number displayed in step 8 above. This step formats the fixed hard disk in drive 0 as a system hard disk. Wait until the formatting operation is completed indicated by an ISIS-II prompt before going to step 11.
11. Press the Interrupt 1 switch on the console control panel. Drive 0 is now the system disk drive.
12. To format the removable cartridge as a non-system disk, enter the following command:

    -FORMAT :F2:NONSYS.DSK<CR>
13. The system is now ready to accept another command from the console.

NOTE

All interrupt 1 and ISIS-II aborts reboot from the system hard disk in drive 0. The system flexible disk in drive 4 is required only for system start-up and reset (RESET switch) operations.

Hard Disk Cartridge Installation

Stabilize the disk cartridge temperature to the disk drive environment before it is installed. Be sure that the disk is clean, then install it as follows (see figure 2-6):

1. Check that the MAIN circuit breaker (CB1) is ON.
2. Check that the START/STOP indicator is not lit.
3. Raise the disk drive cover.
4. Lift the cartridge holddown arms.
5. Separate the dust cover from the disk cartridge by sliding the cover release button to the side and lifting the cartridge handle. Remove and set the dust cover aside.
6. Check that brush indicator slot is aligned with the black line. If not, align it with a coin or similar object.
7. Gently place the disk cartridge onto the spindle hub with head opening toward rear of disk drive.
8. Rotate the cartridge slowly back and forth until it seats into detent.
9. Push the cartridge handle down.
10. Replace the dust cover, open end down, over the disk cartridge.
11. Position the cartridge holddown arms over the cartridge.
12. Close the disk drive cover.
13. Press the START/STOP switch. If the spindle motor does not rotate, disk cartridge is not installed properly (press START/STOP and perform steps 4 through 12 again).
14. If the spindle motor rotates, wait until the READY indicator is lit (65 seconds).

Hard Disk Cartridge Removal

To remove a hard disk cartridge, follow these steps:

1. Check that MAIN circuit breaker (CB1) is on and that the blower motor is on.

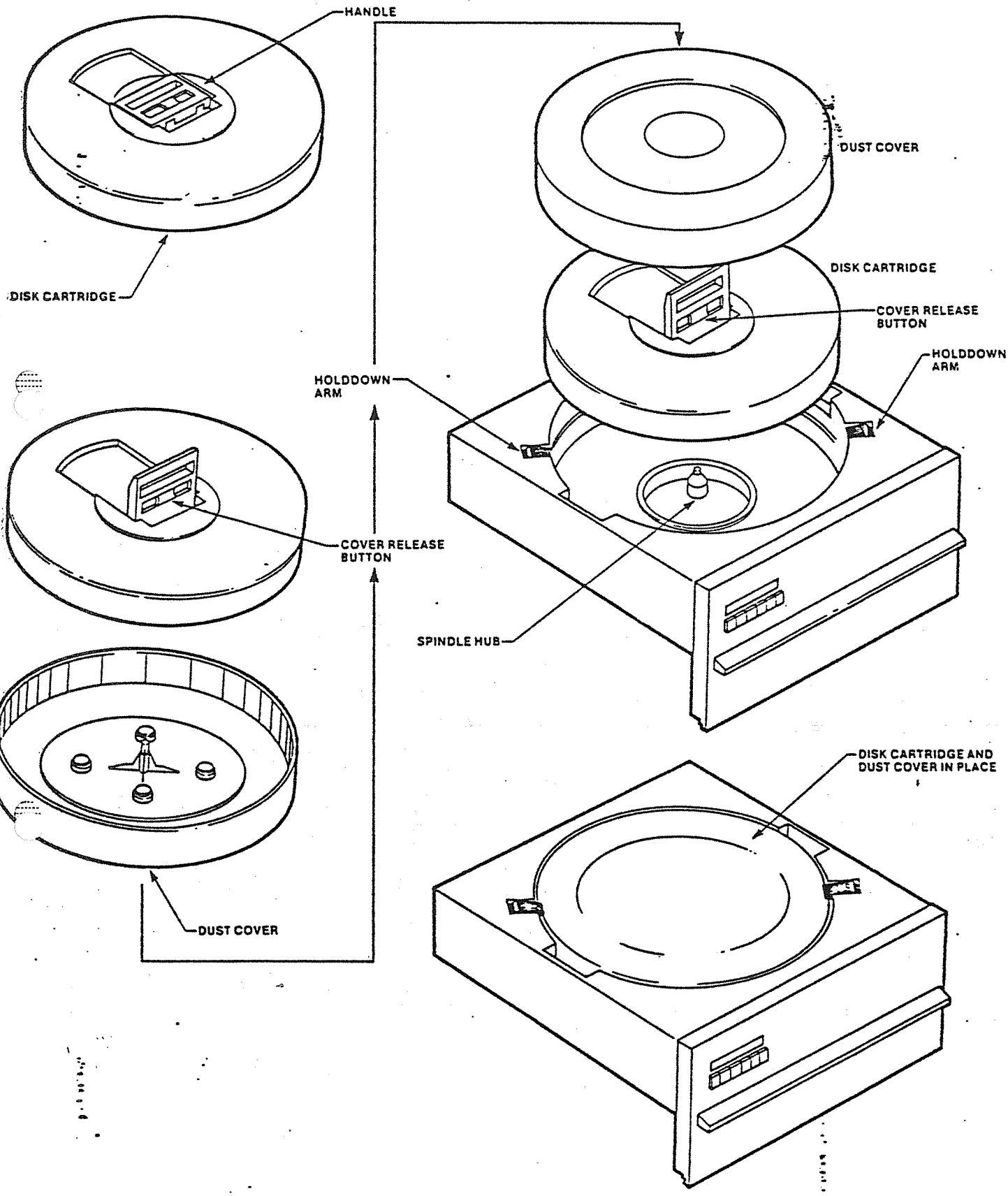


Figure 2-6. Hard Disk Cartridge Installation 943-01

NOTE

If the MAIN circuit breaker is on but the blower motor is not, the cartridge must be removed by a service representative.

2. Check that the START/STOP indicator is not lit.
3. Check that the brush indicator slot is aligned with the black line. If not, align it with a coin or similar object.
4. Raise the disk drive cover.
5. Lift the cartridge holddown arms.
6. Remove cartridge dust cover and set aside.
7. Hold the cover release button, lift the cartridge handle, and lift the cartridge clear of the disk drive spindle.
8. Place the dust cover in position on disk cartridge and release the cover release button.

NOTE

To prevent dust and foreign objects from entering the disk drive, never leave the unit open without a cartridge in place for an extended period of time.

Hard Disk Subsystem FAULT Operation

The FAULT indicator comes on when a nondamaging fault exists, such as when more than one head is selected.

If a momentary power failure occurs, the FAULT indicator does not come on. In such an event, the hard disk heads go into emergency retract and the unit stops. The unit automatically restarts when power returns to normal.

If the FAULT indicator is lit, follow these steps:

1. Check that the system flexible disk is in drive 4.
2. Press the FAULT switch. If the FAULT indicator goes out, and remains out, resume normal operation. If not, continue with step 3.
3. Press the START/STOP switch to remove power from the spindle. Allow spindle to stop, then press the START/STOP switch again. Allow spindle to reach operating speed.

   If the FAULT indicator goes out, resume normal operation. If not, proceed to step 4.
4. Press the START/STOP switch to remove power from the spindle, and contact a service representative.

Hard Disk System Power-down Procedure

To power-down the system, follow these steps:

1. Remove all flexible disks as described under the Flexible Disk Removal section.
2. Press the START/STOP switch. The following occurs:
   - The READY indicator extinguishes.
   - The heads retract.
   - The brush retracts.
   - The START indicator goes out after the spindle stops rotating.
   - The cartridge hold-down arm interlocks open.

CHAPTER 3
DEVICE/FILE MANAGEMENT

You control all device and file input/output functions by means of the ISIS-II console commands described in Chapter 4 and the pathnames (file accessing formats) described in this chapter.

ISIS-II handles disk I/O, and calls on Monitor routines for I/O to other devices.

This chapter describes the following:

- Device/file accessing
- Device names
- Filenames
- Disk directory
- The system console
- Line editing
- Operator controlled pauses
- Interrupting Program Execution

The chapter includes references to:

- Console commands (such as COPY, IDISK and FORMAT) which are described in Chapter 4.
- System calls (such as OPEN, CLOSE and CONSOL) and I/O driver routines, which are described in the *Intellec Series III Microcomputer Development System Programmer's Reference Manual*.

Device/File Accessing

ISIS-II provides a simple but uniform method of identifying each device and each disk file. The assignment of logical names allows you to access system resources without requiring addresses.

The system maintains a record of the device addresses established when the system is configured and a directory on each disk of file addresses. The system converts logical names to true addresses during processing sequences.

When you create a file on a disk, the system enters information about the file into the disk directory. When you delete a file, its entry is deleted from the disk directory.

Device Names

The pathname for a device other than disk files is a system-assigned name in the form:

:device:

ISIS-II assigns names to the following kinds of devices: standard devices, generic devices, and nonstandard devices.

Standard devices are predefined devices for which I/O driver routines are specifically provided. System names assigned to standard devices are:

:F0: through :F9:	Directory on the disk in drive 0 ... 9
:TI:	Teletypewriter keyboard
:TO:	Teletypewriter printer
:TP:	Teletypewriter punch
:TR:	Teletypewriter reader
:VI:	Video terminal keyboard
:VO:	Video terminal screen
:HP:	High-speed paper tape punch
:HR:	High-speed paper tape reader
:LP:	Line printer

The generic devices, system console and byte bucket, do not exist in their own right, but provide flexibility for input or output of your data.

Logical names assigned to the device being used as the system console are:

:CI:	Console input file
:CO:	Console output file

The keyboard and screen are normally :CI: and :CO: respectively. However, you can establish some other device, such as a disk file, as :CI: or :CO: with the CONSOL system call.

You can use the byte bucket for data you do not want saved or displayed. The logical name assigned to the byte bucket is:

:BB:	Byte bucket

Nonstandard devices are those devices for which I/O drivers are not specifically provided. The system names for nonstandard devices are:

:R1:	Paper tape reader 1
:R2:	Paper tape reader 2
:P1:	Paper tape punch 1
:P2:	Paper tape punch 2
:L1:	Line printer 1
:I1:	Console input device
:O1:	Console output device

If you use any nonstandard device in your applications, you must write your own I/O driver routines and define the nonstandard device with the Monitor I/O definition routine. Nonstandard console devices can be assigned to :CI: and :CO: via the CONSOL system call.

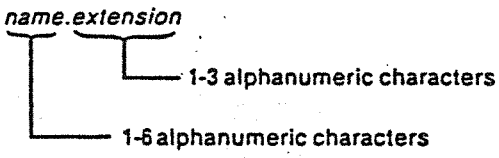
Filenames

All ISIS-II files come with system-assigned names. When you create disk files with a text editor such as CREDIT, you assign a name to that file. The name may be expanded with an extension that further identifies the file.

Filename

The term filename refers to both the name of the file and the extension, if any. Each new file you create must have a unique filename.

The general format for filename is:



where

name is a one- to six-character name you assign to a file. The characters must be alphabetic or numeric.

.extension is a one- to three-character modifier you create for *name*. An *.extension* is optional when the file is created, but if *.extension* is specified, it must always be used when referencing the file.

Default Extensions

Default extensions are predefined extensions that the system assumes under certain programs when you do not supply one. Default extensions are designed to save you time when entering commands.

When you specify any 8086 filename with no extension under the RUN program, the system assumes an extension of .86. If you specify an extension (or a name and period with no extension, as in TEST2.), the default extension is not assumed.

When you specify an 8080/8085 filename with no extension, some ISIS-II programs and the language translators assume a default extension.

Examples of default extensions are:

.OBJ	Translator output
.86	ISIS-II RUN program
.CSD	ISIS-II SUBMIT program
.BAK	ISIS-II CREDIT program
.TMP	ISIS-II LINK, Locate, Lib, CREDIT, and RUN programs
.LNK	ISIS-II LINK program

These extensions are explained further under the individual commands. It is recommended that you do not assign such extensions to your source program's files.

Disk File Pathname

The pathname for disk files is:

:Fn:*filename*

where

:Fn: refers to the directory of the disk in drive n that contains the target file. The value n is an integer between 0 and 9 inclusive. If :Fn: is not specified, :F0: is assumed. *filename* follows :Fn: with no intervening space, as in :F1:MYPROG.

The following example illustrates the file pathname as well as a common use of extensions.

```
:F1:PROGA.SRC    -for the source code
:F1:PROGA.LST    -for the listing from the translator
:F1:PROGA.OBJ    -for the object code
:F1:PROGA.LNK    -for the linked object code
:F1:PROGA        -for the code located at absolute addresses for execution
```

If the disk that contains these files is moved to drive 2, :Fn: becomes :F2:. :Fn: is dependent on the physical location of the disk.

Note that all these files have the same name and are distinguished only by the extension. Extensions allow you to distinguish between different files associated with a single program.

Disk Directory

Files are accessed through a disk directory that keeps track of each file on the disk by its filename.

A directory on a flexible disk has space for 200 entries. A directory on a hard disk has space for 992 entries. This means that a flexible disk can contain up to 200 files, and a hard disk can contain up to 992.

A directory entry contains identifying information about a file. For example, it includes the following items:

- Filename
- Number of blocks allocated to the file
- Number of bytes in the file (length)
- Attributes

Figure 3-1 shows a directory listing obtained by using the DIR command.

```
DIRECTORY OF :F0:X409
NAME   .EXT  BLKS   LENGTH ATTR      NAME   .EXT  BLKS   LENGTH ATTR
ISIS   .DIR    26     3200   IF      ISIS   .MAP     5      512   IF
ISIS   .TO     24     2944   IF      ISIS   .LAB    54     6784   IF
ISIS   .BIN    94    11756  SIF      ISIS   .CLI    20     2407  SIF
ATTRIB         40     4909 WSI       COPY           69     8489 WSI
DELETE         39     4824 WSI       DIR            55     6815 WSI
EDIT           58     7240 WSI       FIXMAP         52     6498 WSI
HDCOPY         48     5994 WSI       HEXOBJ         34     4133 WSI
IDISK          63     7895 WSI       LIB            82    10227 WSI
LINK          105    13074 WSI       LINK   .OVL    37     4578 WSI
LOCATE        120    15021 WSI       OBJHEX         28     3337 WSI
RENAME         20     2346 WSI       SUBMIT         39     4821 WSI
SYSTEM.LIB     24     2849 WS        FPAL   .LIB    74     9125 WSI
FORMAT         62     7789
                                1272
1272/4004 BLOCKS USED
```

Figure 3-1. Directory Listing Example

Filename

Each filename in the directory includes the extension if any.

Blocks

ISIS-II treats a file as a string of bytes. Space is allocated to a file in complete 128-byte blocks even though the last block in the file may be only partially used. In other words, a block is not shared by two files. When you delete a file, the blocks it occupied are released for reassignment by ISIS-II.

For every 62 blocks of data, another block is required for pointers that specify the location of files. You can calculate the number of data and pointer blocks required by a file of N bytes with the following formula:

```
63*(N/128)/62
```

Any result containing a decimal fraction is rounded to the next higher integer. For example, a file of 9000 bytes requires

```
63*(9000/128)/62=
63*70.31/62=
72.15=
73 blocks required
```

Length

The length of a file is the number of bytes contained. Length increases as a file is written and can be affected by other programs containing the system calls OPEN and SEEK.

Attributes

Four attributes are associated with each file that may be set or reset (turned on or off) by the ATTRIB command or the ATTRIB system call:

- Invisible
- Write-protect
- Format
- System

When you create a file with the CREDIT Text Editor, or copy a file with the COPY command, no attributes are set. (The COPY command C switch copies attributes.)

Invisible. The invisible attribute prevents files from being listed by the DIR command unless you use the invisible (I) switch. System files on the system disks supplied with your system have this attribute set.

Write-protect. The write-protect attribute protects files from being modified, deleted, or renamed. However, write-protected files are overwritten by the IDISK and FORMAT commands.

Format. The format attribute protects files from being modified, deleted, or renamed. However, files with the format attribute set are overwritten by the IDISK and FORMAT commands. Files with this attribute set are copied to a new disk by the FORMAT or IDISK commands. The format attribute should be used only with these files: ISIS.DIR, ISIS.MAP, ISIS.T0, ISIS.LAB, ISIS.BIN, and ISIS.CLI, plus ISIS.BAD if you have a hard disk. You should not assign the format attribute to any other file, nor remove the format attribute from those files.

Basic ISIS.* files are reserved for system use. Intel reserves the right to modify or to make these files inaccessible for general use.

System. Files with the system attribute set are copied to a new disk when you specify the S switch with the FORMAT command. When you use the IDISK command, files with the system attribute set are not automatically copied; you can selectively copy them with the COPY command.

The System Console

The keyboard and screen is the 'initial system console.' The console can be changed to another device by a program executing a CONSOL system call or by using the SUBMIT command. The 'current console' is the device currently serving as console, which may or may not be the same as the initial system console.

The console, whatever device it is assigned to, is always the source of system commands. The SUBMIT command directs ISIS-II to take commands from a disk file. The SUBMIT file can return control to the initial system console by means of a CNTL-E. To return control to the SUBMIT file, you enter a CNTL-E from the console.

Under ISIS-II the files :CI: and :CO: are pseudonyms for the devices serving as console input and output. The :CI: file is always the source of system commands. The :CO: file receives console output such as the echo of a command. These two files are always open.

Line Editing

Each character you enter on the keyboard is stored in a line editing buffer until you press the RETURN key or enter a maximum of 122 characters. You can edit or delete the contents of the line editing buffer by using special non-printable editing characters. The line editing characters are:

RUBOUT	Deletes the preceding character from the line editing buffer and from the display. Repeated usage is allowed. (On systems using a teletypewriter, RUBOUT echoes the deleted character to the teletypewriter.)
CNTL-P	Used before an editing character (including itself) to allow entry of the editing character into the line editing buffer.
CNTL-R	Displays the current contents of the line editing buffer.
CNTL-X	Delete the entire contents of the line editing buffer and displays a number sign (#) followed by a carriage return and line feed.
CNTL-Z	Enters an end-of-file and deletes the contents of the line editing buffer and returns 0 bytes of data to caller.

Operator-Controlled Pauses

ISIS-II provides a pause facility that allows you to stop the output, inspect the display or printing, and then resume the output. The special control characters used are:

CNTL-S	Suspends terminal output and delays program execution.
CNTL-Q	Resumes terminal output after the CNTL-S command is given.

These control characters are not line editing characters and have no effect on input operations.

Interrupting Program Execution

Interruption of program execution is described first for the 8080/8085 mode, then for the 8086 mode.

8080/8085 Execution Mode

INTERRUPT switches 0 and 1 on the Intellec front panel terminate processing in 8080/8085 mode. Interrupt 0 returns control to the Monitor whereas interrupt 1 permits ISIS-II to retain control. Interrupt 1 can be used at any time to terminate processing.

When interrupt switch 1 is pressed, the following occur:

- All open files are closed in their current state.
- The initial system console becomes the current console.
- A fresh copy of ISIS-II is read in from the system disk, and ISIS-II prompts for a command (hyphen).

To stop processing and pass control to the Monitor, use interrupt 0. To restart processing of your 8080/8085-programs at the point of interruption, enter a Monitor G command. (Execution starts at the location specified by the contents of the program counter.)

To terminate processing and return to ISIS-II command level from the Monitor, use the G8 Monitor command to close all open files, and restart ISIS-II, which prompts for a command. This is equivalent to pressing interrupt switch 1 except that the Monitor masks all interrupts except interrupt 0, making interrupt switch 1 inoperative when the Monitor is in control.

To pass control from ISIS-II to the Monitor do one of the following:

- Press the interrupt 0 switch.
- Enter the Monitor DEBUG command (described in Chapter 5).
- Execute a LOAD system call with a transfer value of 2 (described in the *Intellec Series III Microcomputer Development System Programmer's Reference Manual*).

To pass control from the Monitor to ISIS-II enter a Monitor G8 command.

8086 Execution Mode

Interrupt 1 performs the same functions in 8086 mode as in 8080/8085 mode.

To interrupt 8086 program execution and enter DEBUG-86, use CNTL-D.

To terminate 8086 program execution and close files, use CNTL-C. CNTL-C returns control to the 8086 RUN program if you are in interactive RUN mode, and to ISIS-II if you are in noninteractive RUN mode. For information on interactive and noninteractive RUN modes, see the RUN command at the end of Chapter 4.

CHAPTER 4
ISIS-II CONSOLE COMMANDS

ISIS-II console commands perform the following basic tasks:

- Prepare a new disk for use by the system (i.e., format the disk)
- Create, delete, and revise files and directories
- Convert object file formats
- Execute your 8080/8085 programs
- Execute your 8086 programs

Command Categories

The following ISIS-II console commands are associated with each of the preceding tasks:

Disk Maintenance Commands

IDISK	Formats a new disk to a basic system or non-system disk.
FORMAT	Formats a new disk and copies files.
FIXMAP	Maps bad sectors on a hard disk.

File Maintenance Commands

DIR	Displays the names of and information about the files listed within the disk directory.
COPY	Copies a file from one device to another.
HDCOPY	Copies one hard disk to another.
DELETE	Removes references to a file from the directory and frees disk storage space associated with that file.
RENAME	Changes the name of a disk file.
ATTRIB	Changes and/or displays the attribute(s) of a disk file.

8080/8085 Program Execution Commands

filename	Loads and executes an 8080 or 8085 program named *filename*
SUBMIT	Enters a file that contains commands to be executed.
DEBUG	Loads an 8080 or 8085 program if specified, and gives control to the Monitor.

8086 Program Execution Commands

RUN — Activates the 8086 execution mode and optionally executes an 8086 program. The following commands control 8086 processing in RUN mode:

- *filename* — Executes an 8086 program named *filename* (see the RUN command).
- DEBUG — Loads an 8086 program if specified, and gives control to DEBUG-86. This command is described in Chapter 6.
- WORK — Changes or displays the default drive used for workfiles.
- DATE — Changes or displays the system date.
- EXIT — Exits the RUN program and returns control to ISIS-II.

File Editing Commands

File creation and editing commands for the CREDIT Text Editor are described in the following manual:

- *ISIS-II CREDIT CRT-Based Test Editor User's Guide*

Program Control and Code Conversion Commands

Program control commands (for Librarian, Linker, and Locator) and code conversion commands (for hexadecimal to/from object module format conversion) are described in the following manuals:

- *MCS-80/85 Utilities User's Guide for 8080/8085-Based Development Systems*
- *iAPX 86, 88 Family Utilities User's Guide for 8086-Based Development Systems*

Entering Commands

Your communication with ISIS-II is through the system console. When you press RESET, ISIS-II signs on and issues its prompt character (a hyphen):

```
ISIS-II Vx.y
-
```

where x.y is the version and release number of ISIS-II.

You can enter a command whenever the ISIS-II prompt is displayed. Each command is entered as a command line and must be terminated by a carriage return or a line feed. When you press the RETURN key after entering a command line, a line feed is automatically entered.

You can intersperse comment lines with command lines. Begin each comment line with a semicolon.

You can correct or edit the current input line with the RUBOUT key. Other line editing characters are described in Chapter 3, Line Editing. Once the RETURN key is pressed that input line can no longer be edited.

The RUN and DEBUG-86 programs are the only ones that execute in the 8086 mode. If the system is in 8086 mode, you must exit to ISIS-II before entering any other ISIS-II command program.

Command Syntax

The general syntax of ISIS-II console command is:

command parameters<cr>

where

command is the name of a command program.

parameters are one or more data required by the command. When you enter more than one parameter, separate them with commas or blank spaces unless noted otherwise under the individual commands. When a parameter consists of switches, they may be separated by spaces, but not by commas.

Parameters enclosed in brackets ([ ]) are optional. If an optional parameter is omitted, default actions are performed by ISIS-II commands as explained with each command.

In most cases a command is executed when the carriage return is encountered. Any exceptions are noted under the individual commands.

Specifying Disk Files

The command syntax of many commands includes the following designation:

:Fn:*filename*

Where this format is shown, the following definitions apply unless otherwise noted under the individual command:

:Fn: refers to the directory of the disk in drive n that contains *filename*. The value n is an integer between 0 and 9 inclusive. If :Fn: is not specified, :F0: is assumed.

filename is the name (and extension if any) of the target file. Enter *filename* immediately after :Fn: with no intervening space, as in :F1:MYPROG.

Disk Maintenance Commands

The commands described in this section are:

IDISK	Formats a disk as a system or non-system disk.
FORMAT	Formats a disk as a system or non-system disk and copies additional files.
FIXMAP	Maps bad sectors on hard disk.

A blank disk must be formatted with either the FORMAT or IDISK command before it can be used on the system. You can format the disk as any of the following:

- A basic system disk containing only the ISIS-II files necessary to start-up and operate the system and maintain the disk file directory.
- A basic non-system disk containing only the ISIS-II files necessary to maintain the disk file directory, leaving more space for data than on a system disk.
- A system or non-system disk containing additional ISIS-II files.

You must use IDISK or FORMAT to convert a non-system disk to a system disk.

If your system has at least two disk drives, use either the FORMAT or IDISK command. If your system has a single disk drive, use the IDISK command.

Use a disk formatted as either a single- or double-density disk only in a disk drive of the same density. If you wish to use that disk in a drive of a different density, format the disk again with FORMAT or IDISK in a drive of the desired density.

Table 4-1 shows the ISIS-II files copied for each FORMAT or IDISK command shown. Note that the COPY command must be used in conjunction with IDISK.

The information in this table applies only if your source disk files have the same file attributes set as those shown in the Attributes column.

NOTE

The F attribute is reserved for the basic format file listed in table 4-1. If you remove the F attribute from those files, your new disk will not be formatted properly. If you assign the F attribute to any other file, that file will not be copied by the FORMAT command.

Table 4-1. Disk Formatting Example

Type of File	File Name	Attributes**	FORMAT	FORMAT A	FORMAT S	IDISK	IDISK S	COPY	COPY S
ISIS-II basic format files	ISIS.DIR	IF	X	X	X	X	X		
	ISIS.MAP	IF	X	X	X	X	X		
	ISIS.T0	IF	X	X	X	X	X		
	ISIS.LAB	IF	X	X	X	X	X		
	*ISIS.BAD	IF	X	X	X	X	X		
ISIS-II basic system files	ISIS.BIN	SIF		X	X		X		
	ISIS.CLI	SIF		X	X		X		
ISIS-II system command files	ATTRIB	WS1		X	X				X
	COPY	WSI		X	X				X
	DELETE	WSI		X	X				X
	DIR	WSI		X	X				X
	•								
	•								
	•								
Notes:			(1)	(3)	(4)	(1)	(2)	(5)	(6)

*ISIS.BAD is supplied with hard disk systems only
**Attributes: I=Invisible F=Format S=System W=Write-protect

(1) Formats a basic non-system disk.
(2) Formats a basic system disk.
(3) Formats a system disk and copies all other files on the source disk.
(4) Formats a system disk and copies all other files that have the S attribute set.**
(5) Copies any files that do not have the S attribute set.
(6) Copies all other files that have the S attribute set.

IDISK—Disk Formatting Command

The IDISK command formats a new disk for use with ISIS-II.

Command Syntax

```
IDISK :Fn:label [switches]<cr>
```

where

:Fn: refers to the directory of the disk to be formatted. The value n is the number of the drive (0-9) where the blank disk is located.

label is the name to be given to the blank disk. The syntax of label is the same as for filename with up to six characters for name and three for extension. *Label* must follow :Fn: with no intervening space or comma, as in :F1:MYDISK.

switches are one or more of the following:

S — Formats the new disk as a basic system disk. If S is not specified, the disk is formatted as a basic non-system disk.

P — Specifies single drive mode. The system prompts for output and system disks, pausing to display the prompt messages and to allow changing of disks. If source and destination drives are the same, the P switch is the default.

FROM n — Specifies the disk drive containing the source disk files needed for formatting the new disk. The value n is an integer 0-9 for drive numbers 0 through 9. If the FROM n switch is not specified, the default is to drive 0. If n is not a valid integer 0-9, an error message results. For example:

```
:F10:MYDISK, INCORRECTLY SPECIFIED FILE
```

Description

IDISK copies only the files needed for a basic disk (whether system or non-system). A basic non-system disk contains only the files needed to format the disk: ISIS.DIR, ISIS.MAP, ISIS.T0, and ISIS.LAB, plus ISIS.BAD if a hard disk. For a basic system disk, IDISK copies two additional files: ISIS.BIN and ISIS.CL1.

If you want other files such as command files copied to the new disk, use the COPY command.

Single Drive Systems. You can use IDISK on single-drive or multiple-drive disk systems. On single drive systems, you are prompted to remove the disk and insert the blank disk. When the formatting is completed, you are prompted to insert the original system disk. See example 1.

Hard Disk Systems. The hard disk platter in drive 0 must be formatted as a system disk. See example 3 below.

When used with a hard disk, IDISK verifies each sector. If IDISK cannot read a sector reserved for an ISIS-II file, the following message is displayed:

```
FATAL BAD SPOT AT LOGICAL ADDRESS (ttt, sss),STATUS=nnnn
```

where ttt is the logical track address (in decimal), sss is the logical sector address (in decimal), and nnnn is the hard disk error status (in hexadecimal).

If the unreadable sector does not correspond to an ISIS-II file, the following message is displayed:

```
BAD SPOT AT LOGICAL ADDRESS (ttt, sss), STATUS=nnnn
```

Since ISIS-II allocates hard disk sectors serially, if no mechanism existed to skip over bad hard disk sectors, the remaining sectors would remain unallocated and unusable. Instead, ISIS-II checks hard disk sectors for irregularities during FORMAT and IDISK operations. If a bad sector is encountered, it is allocated to ISIS.BAD and hard disk formatting continues.

Examples

1. This example formats a new disk in drive 0 as a basic system disk on a single-drive system. IDISK prompts for the new (output) disk and for the system disk. IDISK gives the disk the name SYS.V1. To copy other files on the newly formatted disk, use the COPY command for single drive systems described later in this chapter.

```
-IDISK :F0:SYS.V1 <CR>
SYSTEM DISK
LOAD OUTPUT DISK, THEN TYPE (CR)
LOAD SYSTEM DISK, THEN TYPE (CR)
-
```

2. This example formats a new disk in drive 1 as a basic system disk and gives the disk the name NSYS.V1. The COPY command copies all other non-format files from the disk in drive 0 to the disk in drive 1.

```
-IDISK :F1:SYS.V1 <CR>
SYSTEM DISK
-COPY :F0:* TO :F1:* <CR>
COPIED :F0:ATTRIB TO :F1:ATTRIB
COPIED :F0:COPY TO :F1:COPY
COPIED :F0:DELETE TO :F1:DELETE
.
.
.
-
```

3. This example formats a hard disk platter in drive 0 as a basic system disk and copies the basic files needed to format the disk from the system disk in drive 4. To copy other files onto the newly formatted disk, use the COPY command as shown in the previous example.

```
-:F0:=DISK SYSTEM.DISK :F4: FROM <CR>
```

FORMAT—Disk Formatting Command

The FORMAT command formats a new disk for use with ISIS-II and copies files to the new disk.

To format a disk on a system with a single flexible disk drive, use the IDISK command.

Command Syntax

```
FORMAT :Fn:label [switches]<cr>
```

where

:Fn: refers to the directory of the disk to be formatted. The value n is the number of the drive (0-9) where the blank disk is located.

NOTE

In versions of ISIS-II before version 4.0, :Fn: defaulted to :F1: in the FORMAT command. This default has been removed. If you have SUBMIT files that use this default, you must change them. You will receive an error message if you try to default to :F1:.

label is the name to be given to the disk. The syntax of *label* is the same as for filename with up to six characters for name and three for extension. *Label* must follow :Fn: with no intervening space or comma, as in :F1:MYDISK.

switches are one or more of the following, separated with spaces:

A — Copies all files to the specified disk except files (other than ISIS-II system format files) with the format attribute set. If the source disk is a system disk, the new disk becomes a system disk.

S — Copies the basic format files and all files with the system attribute set. If the source disk is a system disk, the new disk becomes a system disk. (The S switch functions differently under FORMAT than it does under IDISK.)

FROM n — Specifies the disk drive containing the disk files needed for formatting. n is an integer 0-9, for drives 0 through 9. If the FROM n switch is not specified, the default is to drive 0. If n is not a valid integer 0-9, an error message is displayed. For example:

```
:F10:MYDISK, INCORRECTLY SPECIFIED FILE
```

If you specify :F0: and no FROM n switch, or if you specify :F0: and FROM 0, the following error message is displayed:

```
CANNOT FORMAT FROM TARGET DRIVE
```

Description

A disk is formatted as a system or non-system disk depending on the type of source disk used and on the switches specified in the FORMAT command.

When a system disk is formatted, FORMAT copies other files in addition to the basic format files.

When a non-system disk is formatted, FORMAT copies only the basic format files: ISIS.DIR, ISIS.MAP, ISIS.T0, and ISIS.LAB, plus ISIS.BAD if a hard disk.

Hard Disk Systems. The hard disk platter in drive 0 must be formatted as a system disk. See example 3.

When used with a hard disk, FORMAT verifies each sector. If FORMAT cannot read a sector reserved for an ISIS-II file, the following message is displayed:

```
FATAL BAD SPOT AT LOGICAL ADDRESS (ttt, sss), STATUS=nnnn
```

where ttt is the logical track address (in decimal); sss is the logical sector address (in decimal); and nnnn is the hard disk error status (in hexadecimal). If the unreadable sector does not correspond to an ISIS-II file, the following message is displayed:

```
BAD SPOT AT LOGICAL ADDRESS (ttt, sss), STATUS=nnnn
```

Since ISIS-II allocates hard disk sectors serially, if no mechanism existed to skip over bad hard disk sectors, the remaining sectors would remain unallocated. Instead, ISIS-II checks hard disk sectors for irregularities during FORMAT and IDISK operations. If a bad sector is encountered, it is allocated to ISIS.BAD and hard disk formatting continues.

Examples

1. This example creates a duplicate system disk excluding any files that do not have the system attribute set.

```
-FORMAT :F1:ISIS.SYS S<CR>
COPYING SYSTEM FILES
ISIS.TO
ISIS.BIN
ISIS.CLI
.
.
.
SYSTEM.LIB
```

2. This example formats a basic non-system disk on drive 1, giving it the name of LIB.V1. System files are not copied.

```
-FORMAT :F1:LIB.V1
NON-SYSTEM DISK
-
```

3. This example formats a hard disk in drive 0 as a system disk; the files with the system or format attribute set are copied from a system disk in drive 4.

```
-:F4:FORMAT :F0:SYSTEM.DSK S FROM 4<CR>
```

FIXMAP—Hard Disk Mapping Command

The Fixmap command maps bad sectors on hard disk.

Command Syntax

```
FIXMAP drive<cr>
```

where

drive is the number of the hard disk drive on which the command is to operate; *drive* is an integer value of 0 or 1.

Description

The FORMAT and IDISK commands in ISIS-II recognize bad sectors and record the numbers of these sectors to prevent their allocation to files. Fatal errors and disk errors arising during the HDCOPY procedure can also show that sectors are bad; either of the following messages reports that logical track 137, logical sector 106 on drive 1 is bad:

```
STATUS=000F
D=1 T=137 S=106
DISK ERROR-UNABLE TO WRITE TO DESTINATION DISK ON DRIVE 1
LOGICAL ADDRESS (137, 106) STATUS=000F
```

The FIXMAP command records the presence of bad sectors reported in either form illustrated above. There is no corresponding command for flexible disks.

When the FIXMAP command is entered, it displays a sign-on message followed by a FIXMAP prompt (an asterisk):

```
ISIS-II MAP FIXER Vx.y
```

where x.y is the version and release number of the FIXMAP program.

If an earlier malfunction caused damage to a required ISIS-II format file, the following message is displayed:

```
BAD SECTOR COUNT INCONSISTENCY
```

If you use FIXMAP to change the state (good-bad) of any sector, the inconsistency is resolved, but the disk remains unreliable. All important files should be copied to another disk and the unreliable disk reformatted.

FIXMAP Commands

You direct the operation of FIXMAP by entering commands from the following list. FIXMAP prompts with an asterisk whenever it is ready to accept a command.

Mark *disk-address*	Change the known state of a sector from good to bad.
Free *disk-address*	Change the known state of a sector from bad to good.
List *filename*	List all known bad sectors.
Count	List the number of known bad sectors.
Record	Record changes specified by Mark and Free.
Quit	Exit to ISIS-II without recording changes.
Exit	Record changes and exit to ISIS-II.

A command may be truncated at any point after its first character. For example, M, MA, or MAR may be used to stand for Mark.

When a command calls for a disk address, that address should have the form:

track, sector[T]

where

track is a number from 0 to 199 that specifies the logical track address containing the bad sector.

sector is a number from 1 to 144 that specifies the logical sector address of the bad sector within the track.

T is an optional switch indicating that a group of 36 sectors should be processed.

The T switch is appropriate if the STATUS reported in the error message was 0001, 000A, or 000E. (See Chapter 7 for a description of status.)

If the T switch is present, the *sector* number specifies a group of 36 sectors on *track*:

If *sector* is in the range 1-36, that group of sectors is processed.

If *sector* is in the range 37-72, that group of sectors is processed.

If *sector* is in the range 73-108, that group of sectors is processed.

If *sector* is in the range 109-144, that group of sectors is processed.

Track and sector numbers, and the T switch, if present, should be separated by spaces. The track and sector numbers should be those reported in the error message that identified the bad sector.

Mark Command

The Mark command changes the known state of a sector from good to bad.

The syntax of the Mark command is:

MARK *disk-address*<cr>

where

disk-address is the track-and-sector address of the sector to be marked as bad.

If the T switch is present, a group of 36 sectors is marked as bad. A sector known to be bad is not allocated to any file.

If the sector specified in the Mark command is not associated with an existing file, the sector is marked as bad. If the T switch is not present, the system displays:

```
SECTOR MARKED
```

If the T switch is present, no message appears when a single sector is marked; instead, when all 36 sectors have been processed, the system displays the message:

```
TRACK PROCESSED
```

If the sector belongs to an existing file, it cannot be marked as bad. Under any of the following conditions, the sector is not marked:

If the sector belongs to one of the required ISIS-II format files, the system displays:

```
(track,sector) REQUIRED BY ISIS-II
```

The system will be unreliable when the questionable disk is in use. You should format a new hard disk and copy your program and data files onto it.

If the sector is already known to be bad, marking the sector is redundant. The system displays:

```
(track,sector) ALREADY MARKED
```

If the sector belongs to a file other than a required format file, the system displays:

```
(track,sector) IN USE
```

If you know the name of the file, exit to ISIS-II and delete the file; then use FIXMAP to mark the bad sector. If you do not know the name of the file, follow this procedure:

1. Exit to ISIS-II, using either the Quit or the Exit command (described below).
2. Give the command

   -COPY :Fn:*.* TO :Fn: *.* Q<cr>

   where :Fn:*.* refers to the disk containing the source files, and the second :Fn: refers to another hard disk. The Q switch causes the system to query before copying each file. (See the description of COPY.)

3a. *If the copy is successful,* disk :Fn: contains a usable copy of the bad disk. Use FIXMAP to get a list of bad sectors on the bad disk; then use IDISK or FORMAT to reformat that disk, and use FIXMAP to mark any bad sectors missed by the formatting command.

3b. *If an error occurs while the disk is being copied,* write down the last filename displayed by Copy, as well as the track and sector numbers appearing in the error message. Use the Delete command to delete the bad file from the bad disk, then use FIXMAP to mark the bad sector. If an error occurs and prevents you from deleting the file, repeat step 2. The file you attempted to delete will not be copied to the new disk. Repeat step 3a.

Example 1: The following example illustrates the use of the Mark command:

```
*MARK 27,83<cr>
SECTOR MARKED
*MARK 27,83<cr>
(27,83) ALREADY MARKED
```

Free Command

The Free command changes the known state of a sector from bad to good.

The syntax of the Free command is:

```
FREE disk-address<cr>
```

where

disk-address is the track-and-sector address of the sector to be freed for allocation.

If the T switch is present, a group of 36 sectors is freed. You might use this command if you had marked a sector by mistake.

If the sector specified in the Free command is known to be bad, it is freed for allocation. If the T switch is not present, the system displays:

```
SECTOR FREED
```

If the T switch is present, no message appears when a single sector is freed; instead, when all 36 sectors have been processed, the system displays the message:

```
TRACK PROCESSED
```

Under either of the following conditions, the sector is not freed:

If a sector is already free for allocation, freeing the sector is redundant. The system displays:

```
(track,sector) ALREADY FREE
```

If the sector is not free because it is in use by a file, the system displays:

```
(track,sector) NOT A BAD SECTOR
```

There is no reason to free a good sector that is part of an existing file.

Example 2: The following example illustrates the use of the Free command. Note that 8 5 T and 8 10 T identify the same group of 36 sectors, i.e., sectors 1-36 on track 8.

```
*FREE 8,30 5 T<cr>
SECTOR FREED
*MARK 8 5 T<cr>
TRACK PROCESSED
*FREE 8 10 T<cr>
SECTOR FREED
*FREE 8 10 T<cr>
(8,10) ALREADY FREE
TRACK PROCESSED
```

List Command

The List command writes a list of all known bad sectors on the named file.

The syntax of the List command is:

```
LIST [filename]<cr>
```

where

[*filename*] is an optional parameter specifying the listing file.

The listing file may be either an output device or a disk file. It may not reside on the disk being fixed. If no filename is given, the list is printed on the console.

The format of the output is one sector per line, with track and sector numbers separated by a comma. The list includes all sectors marked by FIXMAP, as well as bad sectors found by IDISK and Format.

If there are no known bad sectors, the system displays:

```
NO BAD SECTORS
```

If output is directed to a device other than the console, the following message is displayed after the list is written to the device:

```
LIST WRITTEN
```

If the named file resides on the disk being fixed, the system displays:

```
CANNOT LIST TO TARGET DRIVE
```

Example 3: The following example illustrates the use of the List command. The list is written first to the console, then to a disk file.

```
*LIST.<cr>
180,63
182,115
182,116
182,117
*LIST DISK.FIL<cr>
LIST WRITTEN
```

Count Command

The Count command reports the number of known bad sectors on the disk.

The syntax of the Count command is:

```
COUNT<cr>
```

The command displays the following message on the console:

```
xxxxx BAD SECTORS
```

where

xxxxx is a decimal number, the number of known bad sectors on the disk. A sector that has not been marked, or a sector that has been marked and then freed, is not a known bad sector.

Example 4: The following example illustrates the use of the Count command:

```
*LIST.<cr>
180,63
182,115
182,116
182,117
*COUNT<cr>
4 BAD SECTORS
```

Record Command.

The Record command records the changes specified by Mark and Free.

The syntax of the Record command is:

```
RECORD<cr>
```

When this command is entered, changes specified by Mark and Free are recorded on the disk.

If you intend to use the Exit command to leave the FIXMAP program, the Record command is unnecessary. (Exit is described below.) If you intend to use the Quit command, the Record command is required; otherwise, none of the marking and freeing specified during the work session—or since the last Record command—will actually take effect.

When the recording is complete, the system displays:

```
CHANGES RECORDED
```

If no sector has been marked or freed during the work session—or since the last Record command—the system displays:

```
NO CHANGES
```

Example 5: The following example illustrates the use of the Record command.

```
*RECORD<cr>
CHANGES RECORDED
*RECORD<cr>
NO CHANGES
```

Quit Command

The Quit command stops the operation of FIXMAP and returns to ISIS-II.

The syntax of the Quit command is:

```
QUIT<cr>
```

If the Record command has not been given, changes specified by Mark and Free are not recorded on the disk.

Example 6: The following example illustrates the use of the Quit command. Note that the freeing of sector 12, 86 is not recorded on the disk; therefore, upon reentry to FIXMAP, that sector is still known as bad.

```
*FREE 12,86<cr>
SECTOR FREED
*QUIT<cr>
-:F1:FIXMAP :F3:<cr>
ISIS-II MAP FIXER Vx.y
*LIST<cr>
12,86
```

Example 8: The following example illustrates a typical work session with FIXMAP. You invoke the command and begin by getting a list of all bad sectors on the target drive (drive 1, as indicated in the FIXMAP command). The Count command reports that there are eight bad sectors, and the Record command shows that no sectors have been marked or freed during this work session. You free the last 36 sectors on the track containing track 170, sector 113; all sectors except the eight known bad sectors are already reported to be free. You mark track 170, sector 113 as a bad sector, and again list and count the number of bad sectors. This time, the Record command reports that changes have been made. You free the remaining bad sector, list again, and return to ISIS-II.

```
-FIXMAP :1:<CR>
ISIS-II MAP FIXER V1.0
*LIST<CR>
170,113
170,114
170,115
170,116
170,117
170,118
170,119
170,120
*COUNT<CR>
8 BAD SECTORS
*RECORD<CR>
NO CHANGES
*FREE :170 :F3 :1<CR>
(170,109) ALREADY FREE
(170,110) ALREADY FREE
(170,111) ALREADY FREE
(170,112) ALREADY FREE
(170,121) ALREADY FREE
(170,122) ALREADY FREE
(170,123) ALREADY FREE
(170,124) ALREADY FREE
(170,125) ALREADY FREE
(170,126) ALREADY FREE
(170,127) ALREADY FREE
(170,128) ALREADY FREE
(170,129) ALREADY FREE
(170,130) ALREADY FREE
(170,131) ALREADY FREE
(170,132) ALREADY FREE
(170,133) ALREADY FREE
(170,134) ALREADY FREE
(170,135) ALREADY FREE
(170,136) ALREADY FREE
(170,137) ALREADY FREE
(170,138) ALREADY FREE
(170,139) ALREADY FREE
(170,140) ALREADY FREE
(170,141) ALREADY FREE
(170,142) ALREADY FREE
(170,143) ALREADY FREE
(170,144) ALREADY FREE
TRACK PROCESSED
*MARK :170 :F3<CR>
SECTOR MARKED
*LIST<CR>
170, 113
```

Additional copies of this manual or other Intel literature may be obtained from:

Literature Department
Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051

The information in this document is subject to change without notice.

Intel Corporation makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Intel Corporation assumes no responsibility for any errors that may appear in this document. Intel Corporation makes no commitment to update nor to keep current the information contained in this document.

Intel Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in an Intel product. No other circuit patent licenses are implied.

The following are trademarks of Intel Corporation and may be used only to identify Intel products:

BXP	Intellec	Multibus
CREDIT	iSBC	Multimodule
i	iSBX	PROMPT
ICE	Library Manager	Promware
iCS	MCS	RMX
Insite	Megachassis	UPI
Intel	Micromap	μScope
Intelevision		

and the combination of ICE, iCS, iSBC, iSBX, MCS, or RMX and a numerical suffix.

```
*COUNT<cr>
1 BAD SECTOR
*RECORD<cr>
CHANGES RECORDED
*FREE :F0: 123<cr>
SECTOR FREED
*LIST<cr>
NO BAD SECTORS
*EXIT<cr>
CHANGES RECORDED
```

File Control Commands

The commands described in this section are:

DIR	Lists the entries in the disk directory
COPY	Copies files from one device to another
HDCOPY	Copies files from one hard disk to another
DELETE	Erases filenames from the disk directory
RENAME	Changes the name of disk files
ATTRIB	Changes or displays the attributes of a disk file

Wild Card File Names

The DIR, COPY, DELETE and ATTRIB commands allow you to specify filenames using a wild card construct. Either of two special wild card characters can replace some or all of the characters in a name or extension. The wild card characters mean match anything when the system searches a directory for a filename.

The two wild card characters are:

- An asterisk (*) to specify a wild card match to any number of characters.
- A question mark (?) to specify a wild card match to single character.

The asterisk specifies a wild card match to any name and/or any extension in the directory. For example:

ABC.*—means match any filename with the name ABC and any or no extension.

*.PLM—means match any filename with the extension .PLM, such as A.PLM or MYPROG.PLM.

.—means match all filenames in the directory.

The asterisk can also specify a wild card match for the remainder of the name or extension except for the initial character. For example:

AB*.HEX—means match any filename with AB as the first two characters of the name and HEX as the extension. This example would match: ABC.HEX, ABXYZ.HEX, AB.HEX.

*B.HEX is illegal, since * must follow the initial character.

Each question mark specifies a single character for a wild card match. For example:

A?B.HEX—means match any filename with A and B as the first and third characters of a three-character name and HEX as the extension. This example would match: ACB.HEX, AXB.HEX, AMB.HEX.

A??.*—means match any filename with A as first character of a three-character name and any extension.

:device: cannot include a wild card character.

Wild card constructs enable you to specify multiple filenames with a single reference when using the DIR, ATTRIB, DELETE, and COPY commands. For example, you can copy multiple rough draft files to an output device by specifying:

```
-COPY CHAP?-DFT TO :LP:<cr>
```

DIR—Disk Directory Listing

The DIR command lists the contents of a specific disk directory.

Command Syntax

```
DIR [FOR filename][TO listfile][switches]<cr>
```

The positions of these fields are not fixed.

where

filename is the file (or group of files specified with the wild card construction) whose directory entry is to be listed. If FOR *filename* is omitted, the entire directory is listed. If *filename* is not a wild card name (that is, does not contain * or ?), it is listed even if it has the invisible attribute.

listfile is the name of the file or output device such as :TO: or :HP: to contain the directory listing. If TO *listfile* is omitted, the listing is displayed on the screen.

switches are one or more of the following, separated by spaces:

- 0-9 Lists the directory of the disk in :F0:, :F1:, :F2:, ... :F9:. If omitted, the directory of the disk in drive 0 is listed. If more than one drive number is specified, only the rightmost one has effect. The drive number also overrides any device specification in FOR *filename*.
- I Lists all files, including files with the invisible attribute set. If omitted, only files with the invisible attribute not set are listed.
- F Gives fast output, listing only filenames.
- O Prints the directory in a single column format. The default is double column format.
- Z Prints the number of sectors presently used on the specified disk as a fraction of the number of available sectors.

P Specifies single drive mode. After loading the command, the system pauses with the message:

```
LOAD SOURCE DISK, TYPE (CR)
```

After the source disk is loaded and the RETURN key is pressed, the requested directory is output to the specified device. The system then requests that the system disk be replaced:

```
LOAD SYSTEM DISK, TYPE (CR)
```

Description

The DIR default is the directory output in two columns with the following headings:

```
DIRECTORY OF name.ext

NAME     .EXT   BLKS LENGTH ATTR     NAME     .EXT   BLKS LENGTH ATTR
```

where

name.ext is the label of the disk volume that is assigned by the FORMAT or IDISK command. It has the same syntax as a filename. Each item listed by DIR is explained in the section "Disk Directory" in Chapter 3. The directory listing shows the number of blocks in use and the total number of blocks within the disk (2002, 4004, or 28800).

Examples

1. The following example lists two files of a flexible disk on a single density system. The system files, which have the invisible attribute set, are not listed.

```
-DIR<cr>
DIRECTORY OF :F0:IS00AB.SYS

NAME     .EXT   BLKS LENGTH ATTR     NAME     .EXT   BLKS LENGTH ATTR
PROGA    .HEX     75   9263   W      SUMS              51   6357
                                     126
936/2002 BLOCKS USED
```

2. The following is the same as example 1 except a fast listing is requested:

```
-DIR F<cr>
DIRECTORY OF :F0:IS00AB.SYS
PROGA.HEX
SUMS
936/2002 BLOCKS USED
```

3. The following example requests a directory listing of all format files be sent to the line printer. The format files have the invisible attribute, and ISIS.* is a wild card filename, so the I switch must be specified.

```
-DIR I FOR :ISIS> TO :LP:<cr>
```

4. A single-column fast directory listing of the double-density filexible disk in drive 1 is requested by the following command:

```
-DIR :F1: F D<cr>
DIRECTORY OF DISK :F1:ISI.V10
TYPE.M80
TYPE.HEX
TYPE
1337/4004 BLOCKS USED
```

COPY—Copy a File

The COPY command copies files from one device to another.

Command Syntax

```
COPY [:Fn:]infile [,...] TO {[:Fn:][outfile]} [switches]<cr>
                            {:device:       }
```

where

infile is a file (or group of files when using the wild card construct) to be copied. The copy does not affect the contents of *infile*. If more than one *infile* is specified, they are concatenated in the order specified. When concatenating files, specify the full name and extension of each file. The wild card construct may not be used when concatenating files.

outfile is a file to be created or recreated. If :Fn: is not specified, :F0: is assumed. *outfile* must include the extension, if any. If *outfile* is not specified, :Fn: must be specified.

:device: is an output device, such as :LP:, :TO:, :HP:, or :CO:

switches are one or more of the following:

S Copies files with the system attribute set. For example, the command:

```
-COPY :F0:*.* TO :F1:*.* S<cr>
```

copies only files with the system attribute from drive 0 to drive 1.

N Copies files without the system or format attribute set.

P Specifies single drive mode. When files are to be copied between two disks on the same drive, the system prompts for disk swaps with the following messages:

```
LOAD SOURCE DISK, THEN TYPE (CR)
LOAD OUTPUT DISK, THEN TYPE (CR)
LOAD SYSTEM DISK, THEN TYPE (CR)
```

Q Specifies the query mode. The system displays the following message before a copy is performed: COPY *infile* TO *outfile*? A yes or y response causes the copy to be performed. Any other response causes the copy not to be performed.

C Creates *outfile* with the attributes set from the infile. For example, if file XYZ with the I attribute set is copied to the file ABC, the final file ABC will have the I attribute set.

If this switch is not specified, *outfile* is created with all attributes reset (off). This switch does not copy the format (F) attribute.

B Deletes an existing file without displaying the "ALREADY EXISTS" prompt. The existing file is deleted and recreated with new data.

U Opens *outfile* for update instead of deleting it. The "ALREADY EXISTS" message is suppressed. The length is not changed unless the copy causes an increase in the size of the file.

If U and B are both specified, the U function is performed.

Description

When copying from one device to another, the destination can be disk files or physical devices. The copy must be made from an input device to an output device. For example, you can copy from the reader to the punch but not from the punch to the reader.

If *outfile* is an existing disk file and is not write protected, the following message is displayed:

```
outfile FILE ALREADY EXISTS
DELETE?
```

If you respond to the message with yes or y (followed by a carriage return), COPY deletes the existing file before making the copy. No action is performed if you give any other response.

If *outfile* is write protected, then the following message is output:

```
outfile WRITE PROTECTED
```

Single Drive Mode. The COPY command supports single disk drive systems. You can copy files from one disk to another using only a single drive. The command prompts for the source, output, and system disks as it needs them. If you specify a copy on a drive with no change in filename, the command assumes you want to swap disks and prompts for the swaps. For example, the command

```
-COPY ABC TO ABC<CR>
```

results in prompts to swap disks in drive 0. But the commands

```
-COPY ABC TO B:ABC<CR>
```

and

```
-COPY ABC TO DEF<CR>
```

do not result in prompts for disk swapping. You can also copy files between different disks on the same drive by specifying the P (pause) switch in the command.

Wild Card Designations. When you use wild card designations, the following rules apply:

- Every position in the *infile* that contains an * must have a corresponding * in the *outfile* name.
- Every position in the *infile* name that contains a ? must have corresponding ? or * in the *outfile* name.
- The wild card characters cannot be used in device designations (you cannot specify :F*:).

To selectively copy files with the wild card construct, use the query mode. For example:

```
-COPY B:*.ASM TO B:*.BAK/Q<CR>
```

The system then displays the query message before copying each file.

Copying to Another Disk. The COPY command provides a special case for convenience when copying disk files to a different disk. If *outfile* is to have the same name as *infile*, you need not enter *outfile*. For example:

```
-COPY :F2:ABC.XYZ TO :F2:<CR>
```

is the same as specifying:

```
-COPY :F2:ABC.XYZ TO :F2:ABC.XYZ<CR>
```

This form can be used with wild card designations in *infile*:

```
-COPY :F2:*.* TO :F2:<CR>
```

At the end of the listing of files that were copied the following message is displayed if write-protected files have been encountered:

```
WRITE PROTECTED FILE ENCOUNTERED
```

Possible Error Conditions

If you use a wild card designation when concatenating files, an error message is displayed:

```
-COPY :F1:*.BIC,*.* TO :F0:D<CR>
WILD CARD DELIMITERS DURING CONCATENATE
```

When you use the concatenate operation, *outfile* must not have the same name as *infile*. If it does, the following error message results:

```
-COPY :F1:B TO :F1:B<CR>
SOURCE FILE EQUALS OUTPUT FILE ERROR
```

If the rules governing wild card designations are not followed, the following error message is displayed:

```
-COPY :F1:ABC*.* TO :F0:D<CR>
FILE MASK ERROR
```

Examples

1. This example copies three files to one, overwriting its contents:

```
-COPY :F0:CHAP1,:F0:CHAP2,:F0:CHAP3 TO :F0:BOOK<CR>
:F0:BOOK FILE ALREADY EXISTS,
DELETE? Y<CR>
APPENDED :F0:CHAP1 TO :F0:BOOK
APPENDED :F0:CHAP2 TO :F0:BOOK
APPENDED :F0:CHAP3 TO :F0:BOOK
-
```

2. Example 1 could have been done in the following way:

```
-COPY :F0:CHAP1,:F0:CHAP2,:F0:CHAP3 :F0:BOOK Y<CR>
APPENDED :F0:CHAP1 TO :F0:BOOK
APPENDED :F0:CHAP2 TO :F0:BOOK
APPENDED :F0:CHAP3 TO :F0:BOOK
-
```

3. This example lists a file on the line printer:

```
-COPY :F0:BOOK TO :LP:<CR>
COPIED :F0:BOOK TO :LP:
-
```

4. This example displays a file on the console output device:

```
-COPY :F0:CHAP1 TO :CO:<CR>
(text of CHAP1)
COPIED :F0:CHAP1 TO :CO:
```

5. This example copies a file from the disk in drive 0 to the disk in drive 1:

```
-COPY :F0:PROGA TO :F1:NEWPRG <CR>
COPIED :F0:PROGA TO :F1:NEWPRG
-
```

6. This example copies system files from one disk to another on drive 0:

```
-COPY :F0:ASM80 TO :F0:ASM80 <CR>
LOAD SOURCE DISK, THEN TYPE (CR)
LOAD OUTPUT DISK, THEN TYPE (CR)
COPIED :F0:ASM80 TO :F0:ASM80
.
.
.
LOAD SYSTEM DISK, THEN TYPE (CR)
```

If the files to be copied are quite large (exceeding the size of the available RAM) the LOAD SOURCE and LOAD OUTPUT messages will be displayed more than once. As each file is copied, a COPIED message is displayed. After the last file is copied, the LOAD SYSTEM message is displayed.

7. These examples show valid uses of wild card names with the COPY command:

```
-COPY :F1:*.* TO :F2:<CR>
```
(copy all files except those with the FORMAT attribute)

```
-COPY :F1:A??C TO :F0:D??E<CR>
```

```
-COPY :F1:*.* TO :F3:*.*<CR>
```
(copy all non-system and non-format files)

```
-COPY :F1:A?????.? TO :F0:B?.CPY<CR>
```

HDCOPY—Copy Hard Disk Tracks

The HDCOPY command copies the contents of one hard disk to another hard disk on a track-by-track basis. The data transformed are verified during reading of the data into memory. HDCOPY formats the designation disk before writing data to it.

Command Syntax

```
HDCOPY {indrive TO outdrive} <cr>
       {BACKUP             }
```

where

indrive is the number of the drive containing the source hard disk.

outdrive is the number of the drive containing the destination hard disk.

Both drive numbers must be 0 or 1, but both cannot be the same drive number. Both drives must be a hard disk drive.

BACKUP is a switch that can be used to backup a removable hard disk platter.

Description

If you specify BACKUP, the following actions occur:

- The contents of the disk in drive 1 are copied to the disk in drive 0.
- ISIS-II prompts for the backup disk to be placed in drive 1.
- The contents of the disk in drive 0 are copied to the disk in drive 1.
- ISIS-II prompts for a system disk to be placed in drive 1.
- The contents of the disk in drive 1 are copied to the disk in drive 0.

If a disk error is detected while reading from the source hard disk, the following message appears on the console device, and a sector of 0FFH is written to the destination disk:

```
DISK ERROR-UNABLE TO READ FROM SOURCE DISK ON DRIVE M
     LOGICAL ADDRESS (ttt, sss), STATUS = nnnn
```

If an error is detected when reading from or writing to the destination disk, the following message appears on the console device:

```
DISK ERROR-UNABLE TO WRITE TO DESTINATION DISK ON DRIVE N
     LOGICAL ADDRESS (ttt, sss), STATUS = nnnn
```

In both cases, processing continues. You must decide whether or not to use the destination disk or to attempt to make a new copy (see FIXMAP command).

In these error messages, ttt is the logical track address (in decimal); sss is the logical sector address (in decimal); and nnnn is the hard disk error status (in hexadecimal) as described in Chapter 7.

As each track is copied to the destination disk, a T is printed on the console device. If no errors have been detected during a copy operation, the following message is displayed:

```
VERIFICATION OK
```

If errors have been detected, the following message is displayed:

```
BAD SECTORS ENCOUNTERED
```

Possible Error Conditions

The source and destination drives cannot be the same drive number, or a fatal error results, and an error message is displayed:

```
SPECIFIED DRIVES NOT HARD DISK
```

Exit Command

The Exit command records changes and returns to ISIS-II.

The syntax of the Exit command is:

```
EXIT<cr>
```

The Exit command is equivalent to the Record command followed by the Quit command: changes specified by Mark and Free are recorded on the disk, and control returns to ISIS-II.

Example 7: The following example illustrates the use of the Exit command. (Compare this example with example 6, above.)

```
*FREE :F2: 86<CR>
SECTOR FREED
*EXIT<CR>
CHANGES RECORDED
-FIXMAP :F3:<CR>
ISIS-II MAP FIXER Vx.y
*LIST<CR>
NO BAD SECTORS
```

FIXMAP Error Conditions

The following errors cause immediate termination of FIXMAP and a return to ISIS-II. If execution terminates as a result of one of these errors, work done since the last Record command is not recorded on the disk.

If no hard disk is present, the system displays:

```
USE ON HARD DISK SYSTEM ONLY
```

If no drive number is given in the FIXMAP command, or if an illegal switch is present, the system displays:

```
INVALID SYNTAX
```

If this message appears in response to a command within FIXMAP, it means that the command was typed incorrectly, and it does not terminate the session.

If the specified drive number is greater than 3, the system displays:

```
DRIVE NUMBER OUT OF RANGE
```

(In the maximum configuration of the system, the hard disk drives are numbered 0 and 1.)

If the disk does not exist in the system, is not on-line, or is not properly connected, the system displays:

```
ERROR 30 USER PC xxxx
```

where

xxxx is a hexadecimal number.

Examples:

1. A sample HDCOPY command:

```
-HDCOPY 1 TO 0<cr>
LOAD DISK(S), THEN TYPE (CR)
DRIVE 1 DISK NAME MYDISK.123 IS SOURCE DISK
DRIVE 0 DISK WILL BE OVER-WRITTEN
OK TO CONTINUE?
y<cr>
TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
DRIVE 1 DISK NAME MYDISK.123 COPIED TO DRIVE 0
VERIFICATION OK
HARD DISK COPY COMPLETED
-
```

2. A sample HDCOPY command with the BACKUP switch:

```
-HDCOPY BACKUP<cr>
LOAD DISK IN DRIVE 1, THEN TYPE (CR)
DRIVE 1 DISK NAME MYDISK.123 IS SOURCE DISK
DRIVE 0 DISK WILL BE OVER-WRITTEN
OK TO CONTINUE?
y<cr>
TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
DRIVE 1 DISK NAME MYDISK.123 COPIED TO DRIVE 0
LOAD BACKUP DISK IN DRIVE 1, THEN TYPE (CR)
DRIVE 0 DISK NAME MYDISK.123 IS SOURCE DISK
DRIVE 1 DISK WILL BE OVER-WRITTEN
OK TO CONTINUE?
y<cr>
TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
DRIVE 0 DISK NAME MYDISK.123 COPIED TO DRIVE 1
LOAD SYSTEM DISK IN DRIVE 1, THEN TYPE (CR)
DRIVE 1 DISK NAME ISOOAT.SYS IS SOURCE DISK
DRIVE 0 DISK WILL BE OVER-WRITTEN
OK TO CONTINUE?
y<cr>
TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
DRIVE 1 DISK NAME ISOOAT.SYS COPIED TO DRIVE 0
VERIFICATION OK
HARD DISK COPY COMPLETED
```

DELETE—Delete a Disk File

The DELETE command deletes specified directory entries.

Command Syntax

```
DELETE [:Fn:]filename [Q] [, . . . [Q]] [P]<cr>
```

where

filename is the name of a file to be deleted. The wild card construction can be used to delete a group of files.

Q Specifies the query mode. The system displays the following message before each file is deleted: *filename*, DELETE? A yes or y response causes the deletion. Any other response causes the deletion not to be performed.

P Specifies single drive mode. The system displays prompt messages for disk swaps.

Description

This command effectively removes the specified file or group of files from a disk, making the space it occupied available to ISIS-II for reassignment. A file with the write-protect or format attribute set cannot be deleted.

If *filename* is a file with neither the write-protect nor format attribute set, the file is deleted and a confirming message is sent to the console.

If *filename* does not exist, the following message is sent to the console where *filename* is that specified in the DELETE command:

```
filename, NO SUCH FILE
```

If the file cannot be deleted because it has the write-protect or format attributes set, the following message is sent to the console.

```
filename, WRITE PROTECTED
```

Query Mode. When you use the Q switch, the system displays the query message before deleting each file.

The query mode allows you to selectively delete files when using the wild card construct. For example:

```
-DELETE :F1:SHAP?.?? Q<cr>
```

The system then displays the query message for each file that matches the wild card construct.

Single Drive Mode. If you need to swap disks to delete files, specify the P (pause) switch with the command. Before the deletion is performed, the system displays:

```
LOAD SOURCE DISK, THEN TYPE (CR)
```

When the deletion is completed, the following message is displayed:

```
:Fn:filename, DELETED
LOAD SYSTEM DISK, THEN TYPE (CR)
-
```

Examples:

1. This example deletes three files.

```
-DELETE :F0:CHAP?.*<cr>
:F0:CHAP1.TXT, DELETED
:F0:CHAP2.LST, DELETED
:F0:CHAP3.SRC, DELETED
-
```

2. This example shows an attempt to delete a write-protected file.

```
-DELETE :F0:PROGA.ASM<cr>
:F0:PROGA.ASM, WRITE PROTECTED
-
```

3. This example shows the deletion of a file using the P switch.

```
-DELETE :F0:PROGB.ASM P<cr>
LOAD SOURCE DISK, THEN TYPE (CR)
:F0:PROGB.ASM, DELETED
LOAD SYSTEM DISK, THEN TYPE (CR)
-
```

RENAME—Rename a Disk File

The RENAME command changes the name of a disk file. Only the directory is affected.

Command Syntax

```
RENAME [:Fn:]oldname TO [:Fn:]newname<cr>
```

where

:Fn: must be the same for both *oldname* and *newname*.

oldname is the name of an existing file whose write-protect or format attribute is not set. *oldname* follows :Fn: with no intervening space, as in :F2:MYPROG.

newname is the new name to be assigned to *oldname*. *newname* follows :Fn: with no intervening space, as in :F2:PROG1.

Description

When you enter the command to change the name of an existing file to a new name that does not already exist, the system makes the change in the directory.

However, if another file with the new name already exists, the following message is displayed:

```
newname, ALREADY EXISTS, DELETE?
```

If the existing file is to be deleted, enter a Y or y followed by a carriage return. RENAME will delete the existing file and change the name of *oldname* in the directory.

If the existing file to be deleted is write-protected or if you enter any character other than Y or y, the existing file is not deleted and the file to be renamed is not renamed.

NOTE

RENAME cannot be used on nonsystem disks on a single drive system. To change the name of a nonsystem disk file with only a single drive, use the COPY command to copy the file to a file with the new name, then delete the old file with the DELETE command.

Possible Error Conditions

If *oldname* is a nonexistent file, an error occurs.

If the :Fn: part of *oldname* does not match the :Fn: part of *newname*, an error occurs.

Examples

1. The name of a file on drive 0 is changed from CHAP1 to CHAP.ONE:

```
-RENAME :CHAP1 TO :CHAP.ONE<cr>
-
```

2. An attempt is made to rename a write-protected file:

```
-RENAME :NEWPRG.TXT TO :PROGA.TXT<cr>
NEWPRG.TXT, WRITE PROTECTED
-
```

3. In this example, the new name is the name of an existing file.

```
-RENAME :TEXT.BAK TO :TEXT.OLD<cr>
TEXT.OLD, ALREADY EXISTS, DELETE? Y<cr>
-
```

ATTRIB—Change/Display Disk File Attributes

The ATTRIB command changes and/or displays the specified attributes of a disk file.

Command Syntax

```
ATTRIB [:Fn:]filename [attriblist],[Q]<cr>
```

where

filename is a disk file whose attributes are to be changed. The wild card construction can be used to change and/or display the attributes of a group of files.

attriblist is one or more of the following:

I0 or I1	Resets (I0) or sets (I1) the invisible attribute. When set, the file is not listed by the DIR command unless the I switch is specified in the DIR command.
W0 or W1	Resets (W0) or sets (W1) the write-protect attribute. When set, the file cannot be opened for output or update, and cannot be deleted or renamed.

F0 or F1 Resets (F0) or sets (F1) the format attribute. Removal of the format attribute from system files will cause improper formatting of new system disks. This attribute is reserved for specific system files and should not be assigned to any other file. Assigning this attribute to any other file will cause that file not to be copied by the FORMAT command.

S0 or S1 Resets (S0) or sets (S1) the system attribute. When set, the file is copied to the disk being formatted by the FORMAT command when the S switch is used. This file is also copied by the COPY command when the S switch and wild card notation are used.

If two values of the same attribute are specified, for example both I0 and I1, the one rightmost in the command takes precedence.

Q Specifies query mode operation.

Description

When you specify the Q switch, ATTRIB displays the following messages before changing the attributes of a file:

```
filename, MODIFY ATTRIBUTES?
```

Type a Y or y if you want the file attributes modified. Any other response causes ATTRIB to leave the attributes unchanged for the specified file and to go on to the next file in the group.

If a nonexistent disk file is specified, ATTRIB displays:

```
filename, NO SUCH FILE
```

If a non-disk file is specified, ATTRIB displays:

```
filename, NON-DISK DEVICE
```

When attributes for a file have been changed, the current attributes for the file are displayed.

Examples

1. This example changes the write-protect attribute of a group of files:

```
-ATTRIB :F0:PROGA.* W1 <=>
      FILE          CURRENT ATTRIBUTES
 :F0:PROGA.SRC              W
 :F0:PROGA.OBJ              W
-
```

2. This example sets the system attribute for the TYPE program so it will be transferred onto new system disks (see FORMAT command).

```
-ATTRIB :F0:TYPE S1 <=>
      FILE          CURRENT ATTRIBUTES
 :F0:TYPE.                  S
-
```

Program Execution Commands

You can call a program in three ways:

- Direct execution in which you must respond to any queries from the program and to any errors encountered during program execution.
- Debug execution in which the debugging provisions of the system aid you in identifying and locating program erros.
- Non-interactive execution in which you submit the program as a job to be handled by the system without any interaction on your part. You prepare a file that interacts with the program in the same manner as you would during direct execution of the program (see the SUBMIT command).

Two sets of execution commands are provided, one for your 8080- or 8085-based programs, and another for your 8086-based programs.

8080/8085 Program Execution Commands

The commands described in this section are:

filename	Loads and executes an 8080/8085 program named *filename*.
DEBUG	Loads an 8080/8085 program, if specified, and gives control to the Monitor.
SUBMIT	Enters a file that contains commands to be executed.

Filename—Direct Program Execution

Your 8080/8085-based programs are loaded and executed by simply entering the name of the file. You may include parameters with the filename to provide control over the program to be executed. However, the program must be written to accept these parameters and must read the parameters from the line ending buffer. (Further details on parameters are given in the *Intellec Series III Microcomputer Development System Programmer's Reference Manual*.)

DEBUG—Transfer Control to Monitor

The DEBUG command loads an executable 8080/8085 program, if specified, and passes control to the Monitor.

Command Syntax

```
-DEBUG [[:Fn:]filename [parameters]]<cr>
```

where

:Fn: is the directory on the drive n that contains the target file. n is any integer value between 0 and 9 inclusive. If :Fn: is not specified, :F0: is the default.

filename is any ISIS-II command or the file name of an executable program. The program must be an absolute object module. If *filename* is omitted, control transfers to the Monitor, but no program is loaded.

parameters are the normal parameters of the program to be executed.

Description

When your executable 8080/8085 program is loaded, the Monitor displays the contents of the program counter and prompts for a command with a period (.) on the system console.

To begin execution of the program, enter the Monitor G command. You may specify a starting address (entry point address) and up to two breakpoint addresses in the G command.

When execution of your program is suspended at the breakpoint address, you can use other Monitor commands to inspect and/or change the contents of memory and/or registers and then continue program execution from the point of suspension with another G command.

You can return to ISIS-II from the debug mode and reset the debug switch in one of the following ways:

- Enter the Monitor command G8.
- Execute an EXIT system call in the program being debugged.
- Press interrupt 1.

Examples

1. This example executes a program named LIST in debug mode at a load address of 3680H:

```
-DEBUG LIST FILE.TXT<CR>
#3680

.G<CR>
```

(The LIST program is executed.)

2. This example executes the same program in debug mode, suspends execution at the specified breakpoint address, and then returns to ISIS-II with a G8 command instead of letting the program issue an EXIT system call:

```
-DEBUG LIST FILE.TXT<CR>
#3680

.G,36A0<CR>
```

(Use Monitor commands to examine registers and memory when the breakpoint 36A0 is reached.)

```
.G8<CR>

ISIS-II,Vx.y
-
```

3. This example allows you to transfer to Monitor control with no program loaded. Return to ISIS-II by entering the Monitor G command with no address.

```
-DEBUG<CR>
#0008
```

SUBMIT—Non-Interactive Program Execution

The SUBMIT command causes ISIS-II to take its commands from a disk file rather than the console.

Command Syntax

```
SUBMIT [:Fn:]filename[(parameter[, . . .])9]<cr>
```

where

filename is the name (and extension, if any) of the file containing the command sequence definition (explained below). If extension is omitted, SUBMIT assumes the default extension *.CSD*.

parameter is an actual value that is to replace a formal parameter in the command sequence definition file. The maximum number of parameters allowed is 10. If you omit a parameter from the SUBMIT list, enter a comma in its place.

A parameter is a character string of up to 31 characters. Any ASCII character from 20H to 7AH is legal, except a comma, space, or right parenthesis. If a parameter contains a comma, space, or right parenthesis, enclose the parameter in quotation marks. To use a quotation mark inside a quoted parameter, use two quotation marks in its place. For example:

```
'TITLE(''QUOTE ('') SEARCH ROUTINE'')'
```

is used in the final command as:

```
TITLE('QUOTE (') SEARCH ROUTINE')
```

Description

SUBMIT uses two files:

- A command sequence definition (CSD) file that contains the command sequence definition. You create this file with formal parameters.
- A command sequence (CS) file that contains the command sequence to be executed. SUBMIT creates this file with the actual parameters supplied in your SUBMIT command replacing the formal parameters. The command sequence file has the same name as the command sequence definition file but with the extension CS. You should not modify this file.

  SUBMIT reassigns the console input device to the CS file it has created and returns control to ISIS-II, which then executes the commands in the CS file. The CS file has a final command that restores the console input device to its former device assignment and deletes the CS file.

When you create the CSD file, specify formal parameters by using two characters, %*n*, where *n* is a digit from 0 through 9. You may place formal parameters anywhere in the CSD file. To enter a percent sign (%) that is not to be interpreted as a formal parameter, enclose it in single quotes.

You can execute any program noninteractively that reads its commands from :CI:. To execute an 8086 program under SUBMIT, include the RUN command in the CSD file as shown in example 1 below.

The CSD file can also contain commands to the programs being run. If you use a SUBMIT command in a CSD file, it causes another CS file to be created. You can nest SUBMIT commands to any depth.

A CNTL-E (↑E) in a CS file switches the console input from the CS file to the initial system console, allowing interactive processing. To return control to the CS file, enter CNTL-E at the console. If control is *not* returned to the CS file, or if an error occurs after a command sequence has started processing, control returns to ISIS-II and the CS file is not deleted.

NOTE

If a CS file returns control to the initial console device while entering commands in 8086 mode, CNTL-D and CNTL-C will lose their special meaning, i.e., they will not interrupt processing.

Any program running under SUBMIT must allow two buffers in addition to the open files and buffers required by the program itself. See the *Intellec Series III Microcomputer Development System Programmer's Reference Manual* for information on how to determine the base address of your program.

Examples

1. The following example shows a PL/M-86 compilation executed noninteractively on a 4-drive system. The PL/M-86 command has only three items that change. Using SUBMIT to enter the command automates the process, saving you keystrokes at the console.

   The command sequence definition is in the file PLM86.CSD. See the *iAPX 86, 88 Family Utilities User's Guide for 8086-Based Development Systems* manual for an explanation of the controls in the PLM86 command. The file P86.CSD contains the following:

   ```
   RUN PLM86 :F1:%0.%1 DEBUG XREF PRINT (:F3:%0.LST) DATE(%2)
   ```

   This command sequence definition contains three formal parameters, indicated by %0, %1, and %2. The SUBMIT command used to start the compilation is as follows:

   ```
   -SUBMIT P86(PROGA SRC 9 SEPT 80)
   ```

   The command sequence created and executed by SUBMIT is shown as it would be echoed on the console output device:

   ```
   -RUN PLM86 :F1:PROGA.SRC DEBUG XREF PRINT
     (:F3:PROGA.LST) DATE(9 SEPT 80)

   SERIES III PL/M-86 COMPILER, V1.0
   PL/M-86 COMPILATION COMPLETE 0 PROGRAM ERROR(S)

   -:F0:SUBMIT RESTORE P86.CS(:VI:)
   -
   ```

2. This example shows a PL/M-80 compilation, a LINK, and a LOCATE executed from a SUBMIT file on a 2 flexible disk drive system. A CNTL-E is entered in the command sequence definition after the PL/M compilation so you can remove the compiler disk. When the regular system disk (with LINK and LOCATE) is mounted, you enter CNTL-E to resume processing. The text editor does not echo the ↑E; however, it is echoed when the SUBMIT file is executed.

   The file CMPLNK.CSD in drive 1 contains the following command sequence definition. See the *MCS-80/85 Utilities User's Guide for 8080/8085-Based Development Systems* for an explanation of controls in the PLM80 command. The CMPLNK.CSD file contains:

   ```
   PLM80 %0.%1 DEBUG XREF DATE(%2)
   ↑E
   LINK %0.OBJ,SYSTEM.LIB TO %0.SAT&
   PRINT(%0.MP1) MAP
   LOCATE %0.SAT PRINT(%0.MP2) MAP
   ```

The SUBMIT command entered to compile, link, and locate PROGA.SRC follows:

```
-SUBMIT :F1:CMPLNK (:F1:PROGA.SRC, '3 OCT 81', 'XR2')
```

The command sequence actually executed is shown as it would be echoed on the console output device:

```
-PLM80 :F1:PROGA.SRC DEBUG XREF DATE(3 OCT 81)

ISIS-II PL/M-80 COMPILER V3.1
PL/M-80 COMPILATION COMPLETE 0 PROGRAM ERROR(S)

-↑E↑E
-LINK :F1:PROGA.OBJ,SYSTEM.LIB TO :F1:PROGA.SAT &
**PRINT(:F1:PROGA.MP1) MAP
-LOCATE :F1:PROGA.SAT PRINT(:F1:PROGA.MP2) MAP
-:F0:SUBMIT RESTORE :F1:CMPLNK.CS(:VI:)
-
```

8086 Program Execution Commands

The commands described in this section are:

RUN	Activates the 8086 execution mode and optionally executes an 8086 program.
filename	Loads and executes an 8086 program named *filename* (see the RUN command).
DEBUG	Loads an 8086 program, if specified, and gives control to DEBUG-86. This command is described in Chapter 6.
WORK	Changes or displays the default drive used for workfiles.
DATE	Changes or displays the system date.
EXIT	Exits the 8086 execution mode and returns control to ISIS-II.

RUN—Activate 8086 Execution Mode

The RUN command activates the 8086 execution mode, and optionally loads and executes an 8086 program.

Command Syntax

```
RUN [[:Fn:]filename [parameters][;comments]]<cr >
```

where

filename is the name of your 8086 program. If you enter no extension, the system assumes a default extension of .86. For example, if you enter MYPROG as the filename, the system looks for MYPROG.86. This default extension is not assumed if you enter your own extension (or a period and no extension, as in MYPROG.).

comment is one or more ASCII characters not including a carriage return or line feed. Comments always begin with a semicolon.

Description

When a RUN command requires more than one line, terminate each intermediate line with an intermediate line terminator that consists of an ampersand (&) followed by a carriage return. You can optionally insert a comment, preceded by a semicolon (;), between the ampersand and the carriage return. You can enter up to 120 characters before each line terminator.

When the RUN program is ready to accept a continued command line, it prompts with two angle brackets (>>).

You can enter a comment, preceded by a semicolon, either on a line containing a command or on a separate line. If you enter the comment on the same line as a command, place the comment after the command. If you begin the comment on a new line, use the line only for comments.

Your program can read the filename and parameters in the command line by calling the DQSGETSARGUMENT routine described in the *Intellec Series III Microcomputer Development System Programmer's Reference Manual*.

Operating Modes. You can operate the RUN program in interactive or noninteractive modes.

Noninteractive Mode. In this mode you invoke the RUN program and load and execute your 8086 program by entering one command. After program execution control returns directly to ISIS-II. You must re-enter RUN for each 8086 program to be executed.

Activate this mode as follows:

```
-RUN :F1:MYPROG<cr>
```

Interactive Mode. In this mode you invoke RUN first. RUN signs on; you then load and execute 8086 programs by entering each filename without re-entering RUN. After each program is executed, control returns to RUN.

Activate this mode as follows:

```
-RUN<cr>
```

The Run program signs on and issues its prompt character (>):

```
ISIS-II RUN 8086, Vx.y
>
```

where x.y is the version and release number of the Run program.

You can now execute successive programs by just entering the program name followed by a carriage return. After each program is executed, the RUN prompt character (>) is displayed.

To return to ISIS-II enter the EXIT command (explained later in this chapter).

Interrupting Program Execution. To terminate 8086 program execution and close files and return control to RUN or ISIS-II, use CNTL-C. CNTL-C returns control to the RUN program if you are in interactive RUN mode, and to ISIS-II if you are in noninteractive RUN mode.

To interrupt 8086 program execution and enter DEBUG-86, use CNTL-D.

Interrupt 1 performs the same functions in 8086 mode as in 8080/8085 mode. That is, when interrupt 1 is pressed, all open files are closed in their current state, the initial console becomes the current console, and control of the system is transferred to ISIS-II.

Do not use interrupt switch 0 while in the 8086 mode.

Examples

1. To activate the 8086 execution mode and run one program (noninteractive mode):

```
-RUN :F1:AVG.86<cr>
```

2. To activate the 8086 execution mode and run several programs (interactive mode):

```
-RUN<cr>
ISIS-II RUN 8086, Vx.y
>ASM86 :F1:AVG.86<cr>      (executes ASM86.86)
>:F1:AVG<cr>               (executes AVG.86)
>:F1:AVG.<cr>              (executes AVG)
>:F1:AVG.XYZ<cr>           (executes AVG.XYZ)
>Exit
-
```

WORK—Change/Display Default Drive of Workfiles

The Work command changes or displays the default drive used for temporary workfiles in the 8086 execution mode.

Command Syntax

```
[RUN] WORK [:Fn:]<cr>
```

where

:Fn: specifies the drive n that is to be set as the default drive for your temporary workfiles. n is an integer value between 0 and 9 inclusive. The initial system default is :F1: If :Fn: is not specified, the current default is displayed.

Description

The system assigns filenames in the following form as system workfiles as they are created:

```
nnn.TMP
```

where

nnn are integer values between 000 and 999 inclusive.

If you create files with names of this form on the same disk, they are subject to deletion unless one of the following conditions exists:

- Your files have the write-protect attribute set.
- You use the WORK command to change the default drive being used for your workfiles.

For information on creating temporary workfiles, see the *Intellec Series III Microcomputer Development System Programmer's Reference Manual.*

The default drive is saved in the system file RUN.MAC and is obtained from this file whenever RUN is activated. The RUN.MAC file is assumed to be on the same drive specified for the RUN file. If it does not exist when the WORK command is entered, it is created on this disk.

If the system is already in the RUN program interactive mode, do not enter RUN on the WORK command line.

Examples

1. To change the default drive to drive 0 in the interactive mode:

```
>WORK :F0:<cr>
>
```

2. To display the default drive in the interactive mode:

```
>WORK <cr>
:F0:
>
```

3. To change the default drive to drive 0 in the noninteractive mode:

```
-RUN :F0:WORK :F0:<cr>
-
```

4. To display the default drive in the noninteractive mode:

```
-RUN :F0:WORK<cr>
:F0:
-
```

DATE—Change/Display System Date

The DATE command changes or displays the system date.

Command Syntax

```
[RUN] DATE [nn/nn/nn]<cr>
```

where

nn is any integer value between 00 and 99 inclusive that specifies the date desired. If the date is not specified, the last date entered is displayed. The initial default date is 09/01/80.

Description

You can enter the date in either the U.S. format (month/date/year) or the European format (date/month/year).

The system does not automatically update the date but saves the last date entered in the system file RUN.MAC and obtains it from this file upon each subsequent invocation of RUN. The RUN.MAC file is assumed to be on the same disk specified for the RUN file. If it does not exist when the DATE command is entered, it is created on this disk.

If the system is already in the RUN program interactive mode, do not enter RUN on the DATE command line.

Examples

1. To enter the date December 15, 1981 in the interactive mode:

```
>DATE 12/15/81<cr>
>
```

2. To display the last date entered in the interactive mode:

```
>DATE<cr>
12/15/81
>
```

3. To enter the date March 1, 1981 in the noninteractive mode:

```
-RUN :DATE 03/01/81<cr>
-
```

4. To display the last date entered in the noninteractive mode:

```
-RUN :DATE<cr>
03/01/81
-
```

EXIT—Exit the RUN Program

The EXIT command transfers control from the Run program to ISIS-II.

Command Syntax

```
EXIT<cr>
```

Description

Use the EXIT command to return to ISIS-II after an interactive RUN invocation (i.e., RUN was invoked by specifying RUN followed by a carriage return). You can enter an EXIT command whenever the RUN prompt character (>) is displayed.

Example

To exit the RUN program and return to ISIS-II:

```
>EXIT<cr>
```

CHAPTER 5
THE MONITOR

The Monitor is a program stored in ROM that provides the following supervisory functions:

- Initiation of the system at start-up
- I/O interface to all standard peripheral devices except disks
- Software development of your 8080/8085-based programs

System Initiation

The Monitor handles the initiation of the system when you first start up.

In a system without hard disk drives, the Monitor checks drive 0 for a system flexible disk. ISIS-II files are loaded from disk into memory and control is passed from the Monitor to ISIS-II. If a non-system disk is in drive 0, control is retained by the Monitor.

In a system with a hard disk drive, the Monitor checks drive 4 for a system flexible disk.

After ISIS-II is loaded, you can access the Monitor functions by pressing interrupt 0.

I/O Interface

The Monitor handles I/O interface to the console, printer, paper tape reader, and punch. When an input or output operation to these devices is needed, ISIS-II calls the appropriate Monitor routine. When the operation is completed, the Monitor routine returns control to ISIS-II.

ISIS-II handles disk I/O (see Chapter 3).

8080/8085 Program Development

The Monitor provides a command set that enables you to debug your 8080/8085-based programs. Monitor commands allow you to:

- Display and modify memory and processor registers.
- Initiate execution of your 8080/8085 programs.
- Insert breakpoints into your 8080/8085 programs before execution.
- Read hexadecimal data from an external device into memory.
- Write hexadecimal data from memory to an external device.
- Access user written I/O routines.

Command Categories

The commands described in this chapter are grouped as follows:

Program Execution Commands

Execute (G)	Transfers control from the Monitor to the loaded program and optionally sets one or two breakpoints in the program

Monitor I/O Configuration Commands

Assign (A)	Changes device assignment
Query (Q)	Displays the devices currently assigned

Memory Control Commands

Display (D)	Displays a specified range of memory
Fill (F)	Fills a specified range of memory with a constant value
Move (M)	Copies the contents of a specified range of memory into another area of memory
Substitute (S)	Modifies memory on a byte-by-byte basis

Register Command (X)

Display Form	Displays the contents of all registers
Modify Form	Changes the contents of a single register

Paper Tape I/O Commands

Read (R)	Reads data from paper tape into memory
Write (W)	Writes data from memory to paper tape
End-of-File (E)	Writes an end-of-file record to paper tape
Null Leader/Trailer (N)	Writes null leader and trailer characters to paper tape

Utility Command

Hexadecimal	Add and subtract two hexadecimal numbers

Entering Commands

Your communication with the Monitor is through the system console. When you turn on the system power, the Monitor responds with the following sign-on message and a prompt character (a period) at the left side of the display:

```
SERIES II MONITOR, Vx.y
```

where

x.y is the version and release number of the Monitor.

To load an 8080/8085 program from disk into memory for debugging, enter the DEBUG command described in Chapter 4 specifying the name of the program to be debugged. For example:

```
-DEBUG :F1:MYPROG<CR>
```

Your program is now loaded, control of the system is passed to the Monitor, and the Monitor prompt character (a period) is displayed.

You can enter commands at the console anytime after the Monitor prompt character is displayed at the left margin.

Monitor commands are single alphabetic characters. Many commands have required or optional parameters. Parameters may be alphabetic or numeric as indicated under each command.

Numeric parameters are entered in hexadecimal format. The Monitor recognizes only the numeric characters 0 through 9 and the uppercase alphabetic characters A through F as legal hexadecimal digits. In this manual, hexadecimal numbers are shown with an H appended; however, do not enter the H in a Monitor command.

Normally, commands are executed when you press the return key on the keyboard. Any exceptions to this are fully explained in the individual command descriptions.

Command Syntax

The general syntax of Monitor commands is:

command[*parameters*]<cr>

where:

command is the single alphabetic character for the command.

parameters are one or more variable data supplied with the command. Parameters can be numeric or alphabetic. When a numeric parameter is called for, it must be entered in hexadecimal form and is limited to four hexadecimal digits (0000H through FFFFH). Larger numbers can be entered, but only the four rightmost digits are used by the system. For example, the value 123456H is treated as 3456H by the system.

Where a comma is shown in the syntax, you can use either a comma or a space unless otherwise noted under the individual commands.

Entry Errors

The Monitor checks for several error conditions:

- Invalid characters
- Address value errors
- Checksum errors

Invalid Characters

The Monitor checks the validity of each character entered at the Console device. As soon as it encounters an invalid character, it displays a number sign (#) and aborts the command. It displays the prompt character on the next line and waits for more input. In the following example, 4 is rejected because it is not a valid command:

```
.4#
.
```

The first character entered must be a valid command, otherwise it is rejected by the Monitor.

Address Value Errors

All addresses must be entered in hexadecimal. Any character other than 0-9 and A-F is rejected by the Monitor. In the following example, G is not a valid hexadecimal digit:

```
.D1000,1FFG#
```

Many commands require two addresses where the first address is lower than the second. If the first address is higher than the second, the operation will be performed on the single address specified as the first address. For example, suppose you meant to fill memory from address 900 to address 1000 with the constant FFH but entered the addresses in the opposite order as follows:

```
.F1000,900,FF<CR>
```

The Monitor would place a FF in address 1000H and do nothing else. No indication that an error occurred is given. You will only find the error when you notice that a single byte was filled instead of 100H bytes.

The valid range of address is 0000H through FFFFH. If addresses higher than FFFFH are entered, only the last four digits will be used when the command is executed. For example, if 10000H is entered instead of 1000H:

```
.F10000,900,FF<CR>
```

The command would have been evaluated as:

```
.F0000,900,FF
```

and memory from address 0 through 900 would have been filled with FF. This command would have erased some of the memory that the Monitor itself uses. No indication of this error is given except that the Monitor will not function correctly for some commands until you reboot the system.

Checksum Errors

Object code punched onto paper tape by the Monitor or Assembler contains checksum digits, which permit the Monitor to detect improperly punched tape or a tape reader error condition when the tape is read in.

When the Monitor detects a checksum error when reading from an input device, such as a paper tape reader, it types the number sign (#) and stops reading the tape.

If a checksum error is detected, reread the tape from the beginning. If checksum errors continue, check the tape reader hardware or check the tape for damage.

Program Execution Commands

The following command is described in this section:

Execute (G)	Transfers control from the Monitor to the loaded program and optionally sets one or two breakpoints in the program

G—Execute Command

The Execute command transfers control to the program at the address specified or implied in the command and optionally sets one or two breakpoints in the program to which control is passed.

Command Syntax

```
G[start-address][breakpoint1][,breakpoint2]<cr>
```

where

G is the Execute command code.

start-address is the address to be placed in the program counter. Control of the system is passed to this address. The address must be specified in hexadecimal.

breakpoint1 and *breakpoint2* are points in the program where control is passed back to the Monitor. The breakpoints are entered in hexadecimal.

Description

The Execute command transfers control from the Monitor to your own program. You can specify the starting address and one or two breakpoints with the command. The starting address is optional. If it is not specified in the command, the address in the program counter is used. In the following conditions, you can be sure that the desired address is in the program counter:

- You interrupted your program with the interrupt 0 switch and you have done nothing to destroy the program's registers.
- You loaded the program from paper tape and the end-of-file record contained the entry point address. This entry point address is loaded into the program counter by the Monitor.

- You modified the program counter to your entry point address with the Register command.
- Your program returned control to the Monitor because a breakpoint was encountered.

A breakpoint is the address of an instruction within your program that, if fetched, results in the return of control to the Monitor. Breakpoints allow you to check the contents of registers or data fields in your program. When a breakpoint is reached, the instruction at the address is not executed before control is returned to the Monitor. The instruction at the breakpoint is executed when you return control to your program with the Execute command.

If breakpoints are specified, the Execute command functions differently than most of the Monitor commands. Before the carriage return is entered, the Monitor prompts for breakpoints if a comma is entered after typing G. Command entry is in the following sequence:

1. Enter the command code and, optionally, the start-address followed by a comma:
   .G1FA,
2. The Monitor displays a dash:
   .G1FA,-
3. Enter the first breakpoint address:
   .G1FA,-22C
4. Follow step 5 or 6 depending on how many breakpoints you want to set.
5. If a second breakpoint is not to be set, press RETURN:
   .G1FA,-22C<CR>
6. If a second breakpoint is to be set, enter another comma, enter the second breakpoint, and press RETURN:
   .G1FA,-22C,-25E<CR>

If the command contains a syntax error, no breakpoints are set. The command must be reentered and the breakpoints again specified.

When either of the breakpoints are reached and control is returned to the Monitor, or when control is returned to he Monitor because of an interrupt 0, both breakpoints are eliminated. If you want them when you resume execution of your program, you must specify them again.

There are two important points you must know when using breakpoints to debug and test your program:

- In saving the CPU status for your program, the Monitor uses the top 12 bytes of your program stack. This pushes the status of your registers and program counter onto the stack. You should be aware of this when examining the stack. When control is returned to your program, your registers are restored and the stack pointer is reset as if the breakpoint had never occurred.

- The interrupt system is enabled when the Monitor is entered. The Monitor cannot determine the state of the interrupt system just prior to exit from your program. It is assumed that the interrupt system was enabled and so interrupts are enabled when control is returned to your program. It is your responsibility to either enable or disable the interrupt system.

Examples:

1. To pass control to the program address in the program counter:
   .G<CR>
2. To pass control to the program whose entry pont is 30A:
   .G30A<CR>
3. To pass control to the program whose entry is 30A and to set a breakpoint at address 400 within that program:
   .G30A,400<CR>
4. To pass control to the program whose entry point is 30A and to set two breakpoints, at addresses 400 and 500, within that program:
   .G30A,400,500<CR>

Monitor I/O Configuration Commands

The Monitor has four logical system devices defined:

- Console
- Reader
- Punch
- List

You have the option of selecting the physical device that will perform the required logical device function. The Monitor commands that allow you to control the system I/O configuration are:

Assign (A)	Changes device assignment.
Query (Q)	Displays the devices currently assigned

The characteristics of each logical device and the physical devices that can be assigned to each logical device are as follows:

- The Console is an interactive, character-oriented input and output device. A teletypewriter and a CRT terminal have all these characteristics.
- The Reader is a character-oriented input device that transfers data on command and notifies the calling system when no more data is available. A paper tape reader meets these qualifications.
- The Punch is a character-oriented output device that accepts a character from the calling system and records it on an external medium. A paper tape punch meets these qualifications.
- The List device is a character-oriented output device that accepts a character from the calling program and records it on an external medium in human readable form. A line printer meets these qualifications.

A driver program is required for each physical device assigned. The physical devices for which the Monitor provides driver programs are:

- Teletype console with a keyboard, printer, paper tape reader, and punch. This type of device can be assigned to all the system devices.
- CRT devices with a keyboard that are compatible with the Intellec system. This type of device can be assigned to the Console or the List device.
- High speed paper tape reader. This type of device can be assigned to the Reader device.
- High speed paper tape punch. This type of device can be assigned to the Punch device.

- Line printer. This type of device can be assigned to the List device.
- Batch. This is a non-interactive mode in which CONSOLE input is read from the assigned READER device and written to the assigned LIST device. In preparing a command file for BATCH input, you should enter commands in exactly the same way as if the system were in interactive mode. Each command should end with a carriage return/line feed pair. The period (prompt) character generated by the Monitor in interactive mode should not appear as part of the command. Since the Monitor will continue to read from the READER until the CONSOLE is reassigned, the last command in the BATCH command file should reassign the CONSOLE to prevent the Monitor from reading off the end of the tape.

A—Assign Command

You can assign one physical device to a system with the Assign command.

Command Syntax

```
Alogical-device=physical-device <cr>
```

where

A is the Assign command code.

logical-device is the system device that is to to be assigned a *physical-device*. The possible values for *logical-device* are:

C or CONSOLE
R or READER
P or PUNCH
L or LIST

The equal sign (=) must be entered.

physical-device is the physical device that is to be assigned to the *logical-device*. The possible values of *physical-device* for each *logical-device* are:

Logical Device	Physical Device
CONSOLE	T or TTY (teletype terminal) C or CRT (compatible CRT terminal) B or BATCH (batch mode) 1 (user-defined device for which a user-written program is present)
READER	T or TTY (teletype terminal) P or PTR (high speed paper tape reader) 1 or 2 (user-defined devices for which user-written driver programs are present)
PUNCH	T or TTY (teletype terminal) P or PTP (high speed paper tape punch) 1 or 2 (user-defined devices on which user-written driver programs are present)
LIST	T or TTY (teletype terminal) C or CRT (compatible CRT terminal) L or LPT (line printer) 1 (user-defined device for which a user-written driver program is present)

Examples

1. To assign a high-speed paper tape reader as the system Reader device:

```
.AR=P<cr>
```

or

```
.AREADER=PTR<cr>
```

2. To assign a CRT terminal as the system Console device:

```
.AC=C<cr>
```

or

```
.ACONSOLE=CRT<cr>
```

Q—Query Command

The Query command displays the current status of the system I/O devices. It displays a list of the system devices and the physical devices assigned to them.

Command Syntax

```
Q<cr>
```

where

Q is the Query command code. No parameters are allowed with this command.

Example

To list the current assignments of system devices:

```
.Q<cr>
C=T
R=T
P=T
L=L
```

This response indicates that a teletype terminal is assigned as the Console, Reader, and Punch devices. A line printer is assigned as the *List* device, not the Console device.

Memory Control Commands

There are four Monitor commands for accessing the 8080/8085 memory. The commands that only read memory can be used on RAM as well as PROM and ROM. The commands that write to memory can only effectively be used on RAM. If you specify ROM or PROM with these commands, no error indication is given but the write portion of the command is not executed.

The Memory control commands are:

- Display (D) Displays a specified range of memory.
- Fill (F) Overlays a specified range of RAM with a constant value.
- Move (M) Copies the contents of a specified portion of memory into another RAM location.
- Substitute (S) Modifies RAM on a byte-by-byte basis.

D—Display Command

The Display command displays a section of memory formatted into lines of 16 bytes separated by spaces with the address of the first byte at the left margin.

Command Syntax

```
Dstart-address,end-address<cr>
```

where

D is the Display command code.

start-address is the beginning of the memory range to be displayed. The address must be specified in hexadecimal. The *start-address* must be less than or equal to *end-address*. If *start-address* is equal to *end-address*, a single byte is displayed.

end-address is the end of the memory range to be displayed. The address must be specified in hexadecimal.

Description

If the List device is not ready, the Display command will hang. To continue the operation, make the List device ready, or, if that isn't possible, press the interrupt 0 button on the main chassis.

Example

To display the contents of memory locations 109H through 12AH:

```
.D109,12A<cr>
0109 09 0A 0B 0C 0D 0E 0F
0110 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0120 01 02 03 04 05 06 07 08 09 0A
```

F—Fill Command

The Fill command writes a specified 1-byte constant into a specified RAM area. If ROM or PROM is specified, no error is issued, and the command continues to completion even though the memory does not change.

Command Syntax

```
Fstart-address,end-address,constant<cr>
```

where

F is the Fill command code.

start-address specifies the beginning of the memory range to be filled with the *constant*. The memory address must be entered in hexadecimal.

end-address specifies the last byte of the memory range to be filled with the *constant*. The memory address must be entered in hexadecimal.

constant is the byte to be written to the specified address range. The *constant* must be entered as a hexadecimal number.

Description

If a character other than 0 through F is entered, the command is terminated and the prompt character (.) is displayed.

Example

To initialize memory locations 20H through 2FH with 00H:

```
.F20,2F,00<cr>
```

M—Move Command

The Move command copies a specified area of memory into an area of RAM.

Command Syntax

```
Mstart-address,end-address,destination-address<cr>
```

where

M is the Move command code.

start-address is the address of the first byte to be moved. The address must be specified in hexadecimal.

end-address is the address of the last byte to be moved. The address must be specified in hexadecimal.

destination-address is the address to which the first byte (start-address) is to be moved. Each subsequent byte is moved to a location one higher than the last.

Description

The move is done on a byte-by-byte basis, that is, the first byte of the specified area is copied to the new location, then the second byte is copied to the location following the first new location, and so forth. The data in the original location is not destroyed. Any data existing at the new location is overlaid.

Because the command works on a byte-by-byte basis, you should be careful when attempting to move a block of data to a location within the block. By the time the command reaches the end of the block, the data will have been overlaid by the first data moved.

Examples

1. To move the data currently at address 0100H through 0200H to address 0400H through 0500H:

   ```
   .M100,200,400<cr>
   ```

2. To move the data currently at address 1000H through 1FFFH to address 1500H through 24FFH:

   ```
   .M1500,1FFF,1000<cr>
   .M1000,14FF,1500<cr>
   ```

3. If you tried to do the above example with a single command as follows:

   .M1000,1FFF,1500<cr>

   the first 500H bytes would be copied as you expected, but the second 500H bytes would be a copy of the first 500H because bytes 1500H through 1FFFH were overlaid by the first 500H bytes.

S—Substitute Command

The Substitute command displays memory locations on an individual basis and gives you the option of modifying each location as it is displayed.

Command Syntax

S*address*,[*databyte*][,[*databyte*]][...]<cr>

where

S is the Substitute command code.

address specifies a RAM address. The address must be specified in hexadecimal.

data-byte specifies a single byte of data in hexadecimal that is to replace the byte currently at the location specified by address. This is an optional parameter. If it is not entered, the byte specified by address is not modified.

Description

The Substitute command functions differently than most of the Monitor commands. The function of this command is performed before the carriage return (<cr>) is entered.

The Substitute command functions in the following manner:

1. Enter the command code and the address followed by a comma:

   .S100,

2. The contents of the specified memory location followed by a dash is displayed:

   .S100,FF-

3. You can now do any of the following:

   - Modify the contents of the address by entering a new byte in hexadecimal:

     .S100,FF-AA

   - Look at the next sequential byte of data by entering a comma:

     .S100,FF-,00-

   - End the comma and not modify the data byte by pressing the carriage return key:

     .S100,FF-<cr>

   - Any combination of the first two and finally ending with a carriage return:

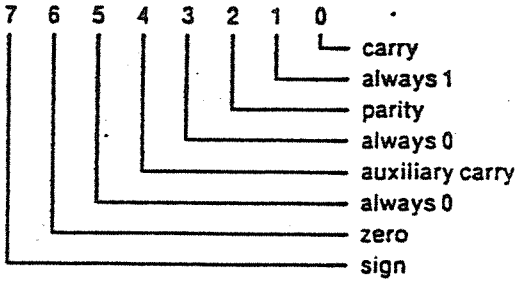
     .S100,FF-AA,00-,11-22<cr>

     The example changes the first byte from FFH to AAH, leaves the second byte unchanged, and changes the third byte from 11H to 22H.

The registers you can display and modify and their symbols in the Monitor are:

Symbol	
A	CPU A register
B	CPU B register
C	CPU C register
D	CPU D register
E	CPU E register
F	CPU flag byte
H	CPU H register
I	Intellec interrupt mask
L	CPU L register
M	CPU H and L registers combined
P	CPU program counter
S	CPU stack pointer

The F register is packed with the CPU condition flags:



X—Register Command (Display Form)

This form of the Register command displays the contents of all the registers. To modify register contents, use the modify form of the command.

Command Syntax

```
X<cr>
```

where

X is the Register command code.

Example

To display the contents of the Intellec registers:

```
.X<CR>
A=00 B=78 C=00 D=47 E=11 F=02 H=FC I=FC L=20 M=FC20 P=1024 S=CD10
```

X—Register Command (Modify Form)

The modify form of the Register command allows you to display and optionally change the contents of the registers, one at a time.

Command Syntax

```
Xregister, [data] [, data]] [,...]<cr>
```

where

X is the Register command code.

register is a single register symbol.

data is one or two bytes of data (depending on the register) to be placed in the register. The data must be entered in hexadecimal.

Description

This form of the Register command functions differently than most of the Monitor commands. The function of the command is performed before the carriage return (<cr>) is entered.

The modify form of the Register command functions in the following manner:

1. Enter the command code and the register symbol:
   .XC
2. The contents of the specified register followed by a dash is displayed:
   .XC FF-
3. You can now do any of the following:
   - Modify the contents of the register by entering new hexadecimal data:
     .XC FF-00
   - Look at the contents of the next sequential register by entering a comma:
     .XC FF-␣EE-
   - End the command and not modify the register by pressing the carriage return key:
     .XC FF-<cr>
   - Any combination of the first two and finally ending with a carriage return:
     .XC ·FF-EE␣EE-␣02-82<cr>
     This example changes the C register from FFH to EEH, leaves the D register unchanged, and changes the flag byte from 02H to 82H.

Examples

1. To examine but not change the M register:
   .XH 1234-<cr>
2. To examine and change the M register:
   .XH 1234-5678<cr>
3. To examine all the registers in sequence and change the D and H registers:
   .XA 00-␣11-␣22-␣33-FE␣44-␣02-␣55-FF␣EC-␣66-␣FF66-␣FC9C-␣E410-<cr>
4. To examine and change the interrupt mask (1) and the L register:
   .XI FE-FF 45-46<cr>

Paper Tape I/O Commands

The Monitor has four commands to support your use of paper tape:

Read (R)	Reads data from a paper tape into the memory.
Write (W)	Writes data from memory to paper tape.
End-of-File (E)	Writes an end-of-file record to paper tape.
Null Leader/Trailer (N)	Writes null leader and trailer characters to paper tape.

The Monitor reads and writes paper tape in hexadecimal format. This format is described in Appendix A.

R—Read Command

The Read command reads a paper tape in hexadecimal format from the device assigned as the Reader and loads the data into memory at the location specified in the record.

Command Syntax

R*bias*<cr>

where

R is the Read command code.

bias is a value (modulo 65,536) to be added to the load address contained in the paper tape record. The data is loaded at the memory location specified by the record address and the bias value. The bias value must be specified in hexadecimal. If there is no bias, a value of 0 must be used.

NOTE

The addition of the bias value does not impiy that the code is relocatable. In some cases, the code would not be executed at the biased location.

Description

The data read is not changed in any way by the specification of a bias value.

Examples

1. To read a paper tape into memory:

   .R0<cr>

2. To read a paper tape into a memory location that is 1000H above the address specified in the tape record:

   .R1000<cr>

W—Write Command

The Write command punches the contents of a specified memory area to the assigned punch device.

Command Syntax

```
Wstart-address,end-address<cr>
```

where

W is the Write command code.

start-address is the first memory location to be punched onto the tape. The *start- address* must be specified in hexadecimal.

end-address is the last memory location to be punched into the tape. *end-address* must be specified in hexadecimal. The *end-address* must be higher than *start-address*.

Description

The Write command does not put an end-of-file record on the paper tape. Thus you can punch non-contiguous areas of memory as a single file. The final record in a paper tape file must be an end-of-file record. After you have written the last memory area to tape, you must write an end-of-file record with the End-of-File command.

Examples

1. To write the contents of memory locations 200H through 3AFH to paper tape:

   ```
   .W200,3AF<cr>
   ```

2. To write the contents of memory locations 450H through 54FH and locations 1000H through 1FFFH to paper tape as a single file:

   ```
   .W450,54F<cr>
   .W1000,1FFF<cr>
   ```

E—End-of-File Command

The End-of-File command punches an end-of-file record in all tapes written by a Write command.

Every paper tape file must have an end-of-file record as the last record. If the end-of-file record is missing, the reader will read off the end of the tape.

Command Sytax

```
Eentry-point<cr>
```

where

E is the End-of-File command code.

entry-point is the entry point address of the program in the file to which the end-of-file record is being added. The *entry-point* must be specified in hexadecimal. A zero should be specified if an entry point address is not wanted.

Description

You can specify an entry point address in the end-of-file record written with the End-of-File command. The entry point address is the address of the first instruction in the program to be executed. When this address is specified in the end-of-file record, the address is loaded into the program counter when the tape is read with a Read command. You can then execute the program by entering a simple Execute (G) command. If the load address field is 0, the program counter is not altered by the Read command.

Examples

1. To punch an end-of-file record in a tape that has just been written by a Write command:

   .E0<cr>

2. To punch an end-of-file record in a tape that has just been written and specify an entry point address to be used when the tape is read with a Read command:

   .E1000<cr>

N—Null Command

The Null command punches a 60 null character leader or trailer.

Command Syntax

```
N<cr>
```

where

N is the Null command code.

Description

The null character is a 00H. You should punch a leader before writing data to a tape and after the end-of-file record. It makes the tape easier to load and saves the data on the tape from the usual damage that tape ends incur through normal handling.

Example

To punch a leader or trailer in a paper tape:

.N<cr>

Utility Command

The utility command performs hexadecimal addition and subtraction.

H—Hexadecimal Command

The Hexadecimal command adds and subtracts two hexadecimal numbers.

Command Syntax

```
Hnumber1,number2<cr>
```

where

H is the Hexadecimal command code.

number1 is the first number to be added. This number is used as minuend for the subtraction. The number must be entered in hexadecimal.

number2 is the second number to be added. This number is used as the subtrahend for the subtraction.

Description

The numbers can include up to four hexadecimal digits.

The command displays two four-digit values. The first is the addition of the two numbers and second is the subtraction. Negative numbers must be entered in their two's complement form.

If more than four digits are entered, the command uses the rightmost four digits. The leading digits are lost.

Example

To add and subtract E49 (minuend) and 111 (subtrahend):

```
.HE49,111<cr>
0F5A 0D38
```

CHAPTER 6
DEBUG-86

DEBUG-86 is a program stored in ROM that provides symbolic debugging of your 8086 programs. DEBUG-86 provides an English language command set that enables you to:

- Initialize DEBUG-86 and load your program from a disk file.
- Specify starting and stopping conditions for program execution.
- Execute your program in real-time mode.
- Execute your program in single-step mode.
- Display and alter 8086 registers, memory locations, and I/O ports.

The files to be debugged by DEBUG-86 must contain executable 8086 object modules. This means that the translator output must be either linked and located to produce an 8086 absolute object module, or linked with the LINK86 BIND control to produce a Position-Independent Code (PIC) or Load-Time Locatable (LTL) object module. These processes are described in the *iAPX 86, 88 FAMILY UTILITIES USER'S GUIDE for 8086-Based Development Systems.*

The development cycle for LTL code is faster because you don't have to locate your source code. The loader locates your code in memory as the code is reached.

Absolute object modules should be located at 7800H.

Command Categories

Utility Commands

DEBUG	Activates DEBUG-86.
EXIT	Exits DEBUG-86.
LOAD	Loads your program code into 8086 memory.

Execution Commands

GO	Causes execution of your program until breakpoint conditions are met.
GR	Sets or displays the contents of the Go register.
STEP	Causes execution of a single program instruction.

Change Commands

Change Register	Changes the contents of a single register or status flag.
Change Memory	Changes the contents of 8086 memory locations.
Change Port	Changes the contents of hardware I/O ports.

Display Commands

Display Registers	Displays the contents of user 8086 registers.
Display Memory	Displays the contents of 8086 memory locations.
Display Memory (ASM form)	Displays the contents of 8086 memory locations in 8086 Assembly language mnemonics.
Display Port	Displays contents of I/O ports.
Display Stack	Displays contents of user's stack.
Display Boolean	Displays boolean value of input.
Evaluate	Displays a value in five number bases.

Symbol Manipulation Commands

Define Symbols	Enters a new symbol in DEBUG-86 symbol table.
Display Symbols	Displays symbols and their values.
Display Lines	Displays statement numbers and their values.
Display Modules	Displays module names.
Change Symbols	Changes value and type of symbols.
Remove Symbols	Removes specified symbols or modules, or all modules, symbols, and statement numbers.
Set Domain	Sets a default module for statement number references.

Compound Commands

Repeat	Causes looping of a command.
Count	Causes looping of a command.
If	Causes execution of a command if a specified condition is met.

Character Set

The valid characters in the DEBUG-86 command language are:

- Upper and lower case alphabetic ASCII characters A through Z
- Digits 0 through 9
- Blank space or comma
- Carriage return/line feed
- $ as a separator in combined words, as in DATASENTRY
- Algebraic operators: +, −, *, /
- Relational operators: =, <, >, <=, >=, < >
- The characters ?, @, &, ;, :, ., (,), +, #, ,, %

All other characters are errors.

Invoking DEBUG-86

You can invoke DEBUG-86 in either of two ways:

- Using the 8086 DEBUG command. This method allows 15k of memory for the DEBUG-86 symbol table, which means space for about 1500 symbols (assuming an average of 10 characters per symbol). However, no I/O buffers are allocated for your program regardless of the number of buffers you may have requested. (None are assumed necessary during debugging.)
- Using CNTL-D to interrupt your 8086 program execution and enter DEBUG-86. This method allows 3k of memory for the DEBUG-86 symbol tables, which means space for about 300 10-character symbols. Buffers are allocated as specified in your program.

Entering Commands

Your communication with DEBUG-86 is through the system console. DEBUG-86 displays an asterisk prompt (*) at the left margin when it is ready to accept a command from the console.

DEBUG-86 takes input from the console but does not interfere with the original command line.

For example:

```
>RUN :F1:DEBUG.PLM86 PROG.SRC PRINT(=LP:) <CR>
```

will load the compiler and transfer control to DEBUG-86. DEBUG-86 does not read the parameters on this command line but goes back to the console for instruction. When the PL/M-86 program is finally executed, the command line parameters are still available to it using DQSGETSARGUMENT (see the *Intellect Series III Microcomputer Development System Programmer's Reference Manual*.)

You can enter up to 120 characters per input line before entering a line terminator (a carriage return or a line feed). On the Intellec terminal, a line feed is automatically entered when you press the RETURN key.

Continuation Lines

A command line can consist of one or more input lines. If you need to continue a line, terminate it with one of the following intermediate line terminators:

- A line terminator embedded in a string that is enclosed in quotes.
- A line terminator preceded by an ampersand (&) that is not embedded in a quoted string or a comment.

  In this case, characters entered between the ampersand and the line terminator are ignored and the ampersand is treated as a space.

When the system is ready to accept a continued line, it displays two asterisks (**).

Comments

You may use comments in any input line. Begin the comment with a semicolon (;). If the line contains commands, place the semicolon and comments after the commands. If you start an input line with a semicolon, use the line only for comments.

If you use an ampersand to continue a command line that also contains comments, place the ampersand before the semicolon. An ampersand that is embedded in a comment is ignored.

Commands are not stored internally. The main use of comments is to document an execution session while it is in progress.

Line Editing

You can use ISIS-II line editing functions to correct errors in the current input line before you enter a line terminator. See Chapter 3 for a description of the line editing characters.

Interrupting Program Execution

You can interrupt 8086 program execution and enter DEBUG-86 by using CNTL-D.

You can terminate 8086 program execution, close files, and return control to RUN by using CNTL-C. If CNTL-C is entered while the system is servicing a system call, a pause may occur before control is transferred.

Interrupt 1 performs the same functions in 8086 mode as in 8080/8085 mode. That is, when you press interrupt 1, all open files are closed in their current state, the initial console becomes the current console, and control is transferred to ISIS-II.

You should not use interrupt 0 while in the 8086 mode.

Error Conditions

For syntax errors, DEBUG-86 displays the input line including the error followed by a number sign (#) and a carriage return.

For all errors, DEBUG-86 displays the following error message:

```
ERROR xxx
```

where

xxx is a decimal error number.

Error messages are defined in Chapter 7.

Expressions

You can use expressions as command arguments to specify numeric values or boolean (true/false) conditions. An expression is a formula that evaluates to a number and represents one of the following;

- Pointer—a pair of 16-bit unsigned integers. The first integer of the pair is called the base and the other integer is called the displacement.
- Integer—a single 16-bit unsigned integer treated modulo 65536, where any bits beyond 16 bits are not used. This is a special case of pointer, with the base value equal to 0.

DEBUG-86 provides only unsigned-integer arithmetic on pointers and integers. Arithmetic operations are applied separately to bases and displacements (i.e., integer arithmetic is always 16-bit). Signed arithmetic is not provided.

The following are examples of expressions.

- Expressions containing a single value:

```
3
FFFFH
```

- Expressions containing operands, operators, and parentheses:

```
2 + 3
174/4
0100H + 00FFH
2 * (6 + 4)
(127 + 44)/20
```

- Expressions containing symbols:

```
.SYMBA - 2
..MOD1.SAM + 21
```

The following sections describe the operands and operators that you can use in expressions.

Operands

You can reference the following types of operands:

- Numeric constant
- Command keywords
- Keyword references
- Register references
- Memory references
- Port references
- Symbolic references
- Statement number references
- String constants

Numeric Constants

A numeric constant produces a fixed unsigned 16-bit integer value, and consists of decimal digits 0 through 9, the letters A through F (hexadecimal digits), and the suffix H for the number base. You do not have to enter the suffix H; it is the default base.

You can specify numbers in other bases by entering the suffix for the new base, as in 100T. The suffix T = decimal base, Q = octal base, and Y = binary base.

The Evaluate command displays constant values in five bases.

Command Keywords

Command keywords are system-assigned names of command functions to be executed. Examples are LOAD, GO, STEP, REGISTER, FLAG, ASM, STACK, and EVALUATE.

Keyword References

Keyword references are system-assigned names of system variables such as registers, status flags, ports and memory. Keyword references allow you to access system variables to display or modify their contents.

You can use keyword references in three ways:

- In an expression, as in RAX+5*8. The keyword value returned is the current contents of the referenced object.
- Alone as in RAX<cr>. The current contents of the referenced objects are displayed.
- On the left side of an equals sign as in RAX = 5555<cr>. The contents of the referenced object is set to the value on the right side.

If the value referenced is less than 16 bits, the system right-justifies the value and fills the unused high-order bits with zeroes. If the value is larger than 16 bits, the extra high-order bits are lost.

Register References

To access a register, use the corresponding keyword reference listed in table 6-1.

The functions of all registers except the pseudo registers are described in *The 8086 Family User's Manual*. The pseudo registers are described under the GO command in this chapter.

Memory References

When you reference a memory location, use the format:

memory-type address

where *memory-type* is one of the following:

BYTE, a one-byte integer value located at *address*.

WORD, a two-byte integer value with low byte at *address* and high byte at *address* + 1.

SINTEGER, same as BYTE.

INTEGER, same as WORD.

POINTER, a four-byte pointer value located at *address* through *address* + 3.

The value of *address* must be an integer.

Table 6-1. 8086 Register Keyword References

Type of Register	Keyword	8086 Register and Interpretation
General Registers	RAX	16-bit Accumulator
	RAH	High 8 bits of Accumulator
	RAL	Low 8 bits of Accumulator
	RBX	16-bit Base Register
	RBH	High 8 bits of Base Register
	RBL	Low 8 bits of Base Register
	RCX	16-bit Count Register
	RCH	High 8 bits of Count Register
	RCL	Low 8 bits of Count Register
	RDX	16-bit Data Register
	RDH	High 8 bits of Data Register
	RDL	Low 8 bits of Data Register
Pointer Registers	SP	16-bit Stack Pointer
	BP	16-bit Base Pointer
Index Registers	SI	16-bit Source Index
	DI	16-bit Destination Register
Segment Registers	CS	16-bit Code Segment Register
	DS	16-bit Data Segment Register
	SS	16-bit Stack Segment Register
	ES	16-bit Extra Segment Register
Status Registers	IP	16-bit Instruction Pointer Register
	RF	16-bit Flag Register
1-bit Flag Registers	AFL	Auxiliary-carry out of low byte to high byte
	CFL	Carry or borrow out of high byte
	DFL	Direction of string manipulation instruction
	IFL	Interrupt-enable (external)
	OFL	Overflow flag in signed arithmetic
	PFL	Parity
	SFL	Sign of the result of an operation
	TFL	Trap used to place processor in single step mode for debugging
	ZFL	Zero indicates a zero value result of an operation
Pseudo Registers	GR	Go register; controls breaking of real-time execution
	BR	Breakpoint registers
	BR0	Breakpoint register 0
	BR1	Breakpoint register 1

When you change memory or reference it in an expression, BYTE is equivalent to SINTEGER and WORD to INTEGER. However, when you display memory, the form of the display is either unsigned (BYTE, WORD) or signed (SINTEGER, INTEGER).

Examples:

```
BYTE 1000H
WORD 101
INTEGER.ABLE
POINTER CS:IP
```

Port Reference

DEBUG-86 supports a maximum of 64K 8-bit (PORT) or 16-bit (WPORT) ports. To reference ports, use one of the following formats, depending on the size of port desired:

```
PORT address
WPORT address
```

Examples:

```
PORT 123
PORT RDX
WPORT 1FFH
```

Symbolic References

DEBUG-86 maintains a symbol table and a source program statement number table that allow you to use symbolic addressing. The DEBUG-86 tables acquire symbols in two ways:

1. When you load your program with the LOAD command, your program symbol information is copied to the DEBUG-86 symbol tables.

2. When you use the Define Symbol command, you enter additional symbols into the symbol table for use during the debugging session.

The number of symbols possible depends on how DEBUG-86 was invoked:

- If you invoke DEBUG-86 with the 8086 DEBUG command, 15k of memory is available for the DEBUG-86 symbol tables. This provides space for about 1500 symbols (assuming an average of 10 characters per symbol.)

- If you invoke DEBUG-86 by interrupting 8086 program execution with CNTL-D, 3k of memory is available for the DEBUG-86 symbol tables. This provides space for about 300 10-character symbols.

A symbol table value is assigned to each symbol. This value represents either the address or the numeric value of the symbol. When you reference a symbol, you are retrieving its address or numeric value.

The following types of symbols and corresponding values are loaded into the tables:

Symbol Type	Value
Instruction and statement labels	Address of the instruction
Program variables	Address of the variable
Program constants	Constant numeric value
Module names	None assigned

In a PL/M-86 or PASCAL-86 program, a module name is the label of a simple DO block that is not nested in any other block. In an Assembly language program, a module name is a label that is the object of a NAME directive.

Symbols contained in a module are local to that module. To reference a symbol name that occurs in different modules, specify the desired module.

The symbol table is organized to preserve the modular structure of your program. Before code is loaded, the table contains a single unnamed module that always comes first. Named modules follow in the order in which they are loaded. Local symbols are stored in the named module in the order of their appearance. Symbols defined without a module name are stored in the unnamed module in the order in which they are defined.

When you enter a symbol reference as an operand, its value is obtained from the table and used in the associated expression. To reference a symbol, use the following format:

[..module].symbol...

If you specify a module name, only that module is scanned; otherwise the entire table is scanned. When you specify more than one symbol, the table is scanned for the first occurrence of each symbol in sequence. The system returns a pointer containing the base and displacement address values for each symbol.

By assigning symbolic names to modules, procedures, and variables, you gain flexibility in retrieving interrelated variables. For example, assume that the symbol .X represents a variable that is used in procedures PROCX, PROCY, and PROCZ of module MODA. You could then retrieve the symbol .X in each procedure as follows:

```
..MODA.PROCX.X
..MODA.PROCY.X
..MODA.PROCZ.X
```

Statement Number Reference

When you reference a statement number, you reference the first instruction generated by the compiler for the source statement. In other words, you are referencing the program location via the statement number.

If different modules (each with its own statement numbers) are linked, or if more than one module is loaded in the DEBUG-86 statement number table, specify a module name in the statement number reference, as follows:

[..module]#statement-number

The statement number is an integer value. If it does not have an explicit suffix, the default suffix is decimal. The value returned is a pointer value that is the absolute address of the first instruction generated by the compiler for the source statement.

Examples:

```
#45
..TEST1#12H
```

String Constants

You can enter any ASCII character (ASCII codes 00H through 7FH) as a string constant by enclosing the character in single quotes. The operand value of a string constant is a 16-bit integer with the high-order bits set to 0, and the low-order seven bits set to the ASCII code. For example, the string constant 'A' has the value 0000000001000001Y (0041H).

Table 6-2 lists the printing ASCII characters and their corresponding hexadecimal codes. (A blank space means a nonprinting character.)

Table 6-2. ASCII Printing Characters and CODES (20H—7EH)

Character	Hex Code	Character	Hex Code	Character	Hex Code
Space (SP)	20	@	40		60
!	21	A	41	a	61
"	22	B	42	b	62
#	23	C	43	c	63
$	24	D	44	d	64
%	25	E	45	e	65
&	26	F	46	f	66
'	27	G	47	g	67
(	28	H	48	h	68
)	29	I	49	i	69
*	2A	J	4A	j	6A
+	2B	K	4B	k	6B
,	2C	L	4C	l	6C
—	2D	M	4D	m	6D
.	2E	N	4E	n	6E
/	2F	O	4F	o	6F
0	30	P	50	p	70
1	31	Q	51	q	71
2	32	R	52	r	72
3	33	S	53	s	73
4	34	T	54	t	74
5	35	U	55	u	75
6	36	V	56	v	76
7	37	W	57	w	77
8	38	X	58	x	78
9	39	Y	59	y	79
:	3A	Z	5A	z	7A
;	3B	]	5B		7B
<	3C	/	5C	I	7C
=	3D	[	5D		7D
>	3E	∧(↑)	5E		7E
?	3F	—(←)	5F		

Operators

An expression can contain any combination of unary and binary operators. A unary operator takes one operand, and a binary operator takes two operands.

Table 6-3 describes the operators you can use with DEBUG-86. The operators are shown as they are entered in expressions and are ranked in order of precedence from highest (1) to lowest (10).

For discussion, the operators are classed as shown in table 6-4.

Table 6-3. DEBUG-86 Operators

Precedence[1]	Operator	Unary Binary[2]	Effect[3]
1	:	u	Base, displacement integer connector for a pointer (e.g., 1234:5678 or CS:IP).
2	+	u	Unary plus.
	-	u	Unary minus; (-N) means (65536-N), the 2's complement of N, modulo 2^{16}
3	*	b	Integer multiplication.
	/	b	Integer division. The result is the integer quotient; the remainder (if any) is lost.
	MOD	b	Modulo reduction. The remainder after division, expressed as an integer.
4	+	b	Addition.
	-	b	Subtraction.
6	content-operator[4]	u	Treats operand as memory or port address, returns the content of that address.
7	=	b	Is equal to. Result is either TRUE (FFFFH) or FALSE (0).
	>	b	Is greater than. Result is TRUE or FALSE.
	<	b	Is less than. Result is TRUE or FALSE.
	<>	b	Is not equal to. Result is TRUE or FALSE.
	>=	b	Is greater than or equal to. Result is TRUE or FALSE.
	<=	b	Is less than or equal to. Result is TRUE or FALSE.
8	NOT	u	Unary Logical (1's) complement. Bitwise 1 becomes 0, 0 becomes 1; TRUE becomes FALSE, FALSE becomes TRUE.
9	AND	b	Bitwise AND. If *both* corresponding bits are 1's, result has 1 in that bit; else 0. TRUE AND TRUE yields a TRUE result; any other combination is FALSE.
10	OR	b	Bitwise inclusive OR. If *either* corresponding bit is a 1, result has 1 in that bit; else 0. If either operand is TRUE, result is TRUE; else FALSE.
	XOR	b	Bitwise exclusive OR. If corresponding bits are different, result has 1 in that bit; else 0. If one operand is TRUE and the other is FALSE, result is TRUE; if both are TRUE or both are FALSE, result is FALSE.

Notes:

[1]1 = highest precedence (evaluated first), 10 = lowest precedence.

[2]u = unary, b = binary.

[3]Refer to text for additional details.

[4]content-operator is one of the tokens BYTE, WORD, SINTEGER, INTEGER, POINTER, PORT, or WPORT.

Table 6-4. Classes of Operators

Class	Operators
(Numeric) Arithmetic unary binary Content unary	 +, –, *, /, MOD, +, –, : *content-operators*
(Boolean) Relational binary Logical unary binary	 =, >, <, <>, >=, <= NOT AND, OR, XOR
Unary	+, –, *content-operators*, NOT
Binary	*, /, MOD, +, –, :, *relational-operators*, AND, OR, XOR

Relational Operators

A relational operator calls for a comparison of its two operands. The six relational operators are shown in table 6-4. Each comparison evaluates to a result that is true (FFFFH) or false (0).

Arithmetic Operators

The DEBUG-86 scanner distinguishes unary "+" and "–" from binary "+" and "–" by context. (Unary "+" is superfluous, since it is the default.)

Unary "–". A unary "–" applied to an integer means 2's complement modulo 65536. In other words, (–N) evaluates to (65536–N). As DEBUG-86 uses only unsigned arithmetic, unary "–" does not apply to pointers.

Binary "+". Binary "+" applies to pointer and integer values only and results in the arithmetic sum of its two operands.

When you add two integers, the result is treated modulo 65536 (any high-order bits after the sixteenth bit are dropped).

When you add a pointer and an integer, the displacement value of the pointer is summed with the integer modulo 65536 and the base value of the pointer is unchanged.

Binary "–". Binary "–" applies to pointer and integer values only and results in the arithmetic difference to the two operands.

When you subtract an integer from another integer, the result is the 2's complement difference. The result is treated modulo 65536, so that a negative result (–N) becomes (65536–N).

When you subtract an integer from a pointer, the result is the 2's complement difference of the pointer displacement and the integer modulo 65536, and the base value of the pointer remains unchanged.

When you subtract a pointer from another pointer (they must have the same base), the result is the 2's complement difference of the displacements modulo 65536, and the resulting base value is set to 0. An error occurs if the base values of the pointers are different.

Binary "*", "/", and "MOD". These operators apply only to integer operands and return only integer results.

Binary "*" results in the multiplication of two integer operands, truncated to the low-order 16 bits.

Binary "/" causes the first integer operand to be divided by the second. The result is the integer quotient; the remainder, if any, is lost. Thus, (5/3) evaluates to (1).

Binary "MOD" returns the remainder after integer division as an integer result, and the quotient part of the division is lost. Thus, (5 MOD 3) evaluates to (2), the remainder of (5/3).

Content Operators

Content operators are keywords that refer to the contents of memory locations and I/O ports. In expressions they function as unary operators with precedence immediately below subtraction. Table 6-5 summarizes the content operators.

To be used in an expression, a content operator must precede a single operand that is a valid address. A range of addresses (using a keyword such as TO or LENGTH) cannot be used in an expression.

Table 6-5. Content Operators

Operator	Content Returned
BYTE	1-byte integer value from the addressed location in user memory.
WORD	2-byte integer value from the addressed location in user memory.
SINTEGER	Same as BYTE.
INTEGER	Same as WORD.
POINTER	4-byte pointer value from the addressed location in user memory.
PORT	1-byte value from addressed 8-bit I/O port.
WPORT	2-byte value from addressed 16-bit I/O port.

Logical Operators

The four logical operators and corresponding examples are shown in table 6-6.

The logical operator NOT results in a 1's complement of an operand; a 16-bit operand is assumed.

The logical operators AND, OR, and XOR each compare a pair of bits. Each operator causes different results as shown in table 6-6.

Table 6-6. Logical Operators

Operator	Bit Comparison			Examples	
	Bit 1	Bit 2	Result	Operation	Result
NOT	1 0	N/A N/A	0 1	NOT 0 NOT 1 NOT 11110110Y NOT FFFFH NOT FFFEH	FFFFH FFFEH 1111111100001001Y 0 1
AND	0 0 1 1	0 1 0 1	0 0 0 1	0 AND 0 1010Y AND 1001Y FFFFH AND 0 FFFFH AND FFFFH 1 AND 0	0 1000Y 0 FFFFH 0
OR	0 0 1 1	0 1 0 1	0 1 1 1	0 OR 0 1 OR 0 1010Y OR 1001Y FFFFH OR 1 FFFFH OR FFFFH	0 1 1011Y FFFFH FFFFH
XOR	0 0 1 1	0 1 0 1	0 1 1 0	0 XOR 1 XOR 0 1010Y XOR 1001Y FFFFH XOR 0 FFFFH XOR FFFFH	0 1 11Y FFFFH 0

Arithmetic and Logical Semantic Rules

Table 6-7 provides a summary of the semantic rules that apply to arithmetic and logical operations. The table specifies the function performed by each type of arithmetic and logical operation, the input required, and the result of the operation (output).

Table 6-7. Arithmetic and Logical Semantic Rules

Operation	Operators	Input	Output	Function
logical	AND, OR, XOR	2 integers	integer	Bitwise conjunction or disjunction of integers.
not	NOT	integer	integer	One's complement of an integer.
relational	<>, <=, >=, <>, =	• 2 pointers, same base • 2 integers	integer integer	Logical test of relational expression. If the displacement integer values satisfy relational operation (true), then the output integer value = FFFFH. If the displacement integer values do not satisfy the realtional operation (false), the output integer = 0. If the base values of the input pointers are not equal, an error occurs.
arithmetic	*, /, MOD	2 integers	integer	Unsigned product (*), quotient(/), or remainder (MOD) of two integers.
memory-content	BYTE, WORD, INTEGER, SINTEGER	pointer or integer	integer	Fetches content of memory location addressed by input value.
memory-content	POINTER	pointer or integer	pointer	Fetches content of memory location addressed by input value.
I/O-content	PORT, WPORT	integer	integer	Fetches content of I/O port (8-bit or 16-bit) addressed by input value.
+(binary)	+	• pointer, integer • integer, integer	pointer integer	Sum of the displacement values, same base as the pointer. Sum of the integers.
-(binary)	-	• 2 pointers with = base values • pointer & integer	integer pointer	Two's complement difference of displacement values. Error occurs if base values are unequal. Two's complement difference of pointer displacement value and integer input, same base as the pointer.
-(unary)	-	integer	integer	Two's complement of the input integer.
+(unary)	+	All types	same	No change.
override base/ construct pointer	:	• integer & pointer • 2 integers	pointer pointer	Replaces current base value of pointer with input integer value. Constructs new pointer with base value set to first input integer value and displacement set to second integer.

Command Contexts

All expressions produce numeric values as results. The interpretation or use of the result depends on the command that contains the expression. Contexts that control the interpretation of an expression are summarized in table 6-8.

A numeric expression is an expression in a numeric command context which treats the result as a numeric value. All bits are significant.

A boolean expression is an expression in a boolean command context. Only integer values may be used in boolean contexts. The least significant bit (LSB) of the result is tested to obtain a TRUE or FALSE value. The result of a boolean expression is TRUE if its LSB is 1, FALSE if its LSB is 0.

A boolean expression uses relational and logical operators to manipulate TRUE/FALSE values. When a relational operator is evaluated, the result is either 0 (FALSE) or FFFFH (TRUE). These results can have a numeric interpretation, but relational operators have limited usefulness in numeric contexts.

When logical operators are applied to TRUE/FALSE values, the results are also boolean, as follows:

NOT: .NOT FALSE → TRUE
NOT TRUE → FALSE

AND: TRUE AND TRUE → TRUE
TRUE AND FALSE → FALSE
FALSE AND TRUE → FALSE
FALSE AND FALSE → FALSE

OR: TRUE OR TRUE → TRUE
TRUE OR FALSE → TRUE
FALSE OR TRUE → TRUE
FALSE OR FALSE → FALSE

XOR: TRUE XOR TRUE → FALSE
TRUE XOR FALSE → TRUE
FALSE XOR TRUE → TRUE
FALSE XOR FALSE → FALSE

Table 6-8. Command Contexts

Type of Entry	Contexts	Interpretation	Limitations	Examples of Use
Numeric expression	Set and change commands, etc.	16-bit unsigned number; bit size may be reduced to fit destination.	All operands and operators allowed. Numeric constant without suffix is interpreted in hexadecimal radix.	IP'= .AA*256T + 10FFFH
Boolean expression	BOOL, IF, UNTIL, WHILE	LSB = 0 → FALSE LSB = 1 → TRUE	All operands and operators allowed. Numeric constants without suffix are interpreted in hexadecimal radix.	.AA AND .BB AND NOT .CC
Address	FROM, content-operator, partition	Pointer to memory or 16-bit (or fewer) address in memory or I/O.	Only arithmetic operators are allowed outside of the outermost parentheses. Constant without suffix are interpreted in hexadecimal radix.	GO FROM .BB + 10
Decimal number	statement-number	positive number	No operators are allowed outside the outermost parentheses. All constants without suffix are decimal.	

Utility Commands

DEBUG-86 utility commands provide file management capabilities. The utility commands are:

DEBUG	Activates DEBUG-86.
EXIT	Exits DEBUG-86.
LOAD	Loads user program code into memory.

DEBUG—Transfer Control to DEBUG-86

The Debug command activates DEBUG-86.

Command Syntax

```
[:Fn:]RUN DEBUG [[:Fn:]filename [parameters]] <cr>
```

where

RUN is a command keyword that invokes the 8086 execution mode.

DEBUG is a command keyword that invokes DEBUG-86.

filename is the name (including extension) of a program that is a valid absolute, PIC, or LTL 8086 object module. If no extension is specified, RUN adds an extension of .86. If filename ends with a period (as in MYPROG.), the null extension is assumed.

parameters is a series of one or more ASCII characters (separated by commas or spaces) representing variable data required by the user program and to be processed by the program.

Abbreviation

None

Description

You can operate DEBUG-86 in either interactive or noninteractive mode.

Noninteractive Mode. Specify DEBUG and optionally your program on the RUN activation line, as follows:

```
-[:Fn:]RUN DEBUG [[:Fn:]filename[(parameters)]]<cr>
```

DEBUG-86 signs on, then prompts for a command with an asterisk (*):

```
DEBUG 8086, Vx.y
*
```

where x.y is the version and release of the DEBUG-86 program.

When you issue an EXIT command after the debugging session, control returns directly to ISIS-II.

Interactive Mode. Specify the RUN program only on the RUN activation line as follows:

```
-[:Fn:]RUN<cr>
```

RUN signs on, then prompts for a command with an angle bracket:

```
ISIS-II RUN 8086, Vx.y
>
```

where x.y is the version and release number of the RUN program.

Issue the DEBUG command and optionally specify your program as follows:

```
DEBUG [[:Fn:]filename[(parameters)]]<cr>
```

DEBUG-86 signs on, then prompts for a command with an asterisk.

When you issue the EXIT command after the debugging session, control returns to the RUN program which displays its prompt (>). The system remains in the RUN interactive mode, and you can execute or debug another 8086 program.

To return control to ISIS-II, issue an EXIT command to the RUN program.

Loading Your Program. You can load the program to be debugged in either of two ways:

Noninteractively, by specifying the program on the DEBUG command activation line. DEBUG-86 signs on after the program is loaded.

Interactively, by specifying only DEBUG on the command activation line. DEBUG-86 signs on without loading your program. You then load your program with the LOAD command explained later in this section. Use this method only if there are no parameters to be passed to your program on the command line. If you use this method, you must enter the entire filename including the default extension .86 that is added by RUN. This method also loads the DEBUG-86 symbol tables.

Examples

1. To activate the 8086 interactive execution mode:

```
-RUN<cr>
ISIS-II RUN 8086, Vx.y
>DEBUG<cr>
DEBUG 8086, Vx.y
*
```

2. To transfer control directly to DEBUG-86:

```
-:F1:RUN DEBUG<cr>
DEBUG 8086, Vx.y
*
```

3. To load your 8086 program and transfer control to DEBUG-86 in one step:

```
-RUN DEBUG :F1:PROG2.86<cr>
DEBUG 8086, Vx.y
*
```

4. To transfer control directly to DEBUG-86 then load your program with the LOAD command:

```
-RUN :F3:DEBUG<cr>
DEBUG 8086, Vx.y
*LOAD :F3:PROG2.286<cr>
*
```

EXIT—Exit DEBUG-86

The EXIT command exits DEBUG-86.

Command Syntax

```
EXIT<cr>
```

where

EXIT is a command keyword that transfers control from DEBUG-86 to the RUN program in the interactive mode, or to ISIS-II in the noninteractive mode.

Abbreviation

EXIT can be abbreviated to EXI.

Description

In the noninteractive mode, the EXIT command operates as follows:

```
-RUN :F3:DEBUG<cr>
DEBUG 8086, Vx.y
*EXIT<cr>
-
```

In the interactive mode, the EXIT command operates as follows:

```
-RUN<cr>
ISIS-II RUN 8086, Vx.y
>DEBUG<cr>
DEBUG 8086, Vx.y
*EXIT<cr>
>EXIT<cr>
-
```

The last EXIT command is the RUN EXIT command (see Chapter 4, ISIS-II Console Commands).

For further information on interactive and noninteractive modes, see the DEBUG command explained on the previous page.

Possible Error Conditions

None, except for possible ISIS-II errors that may occur when the system is in noninteractive mode and you are returning to ISIS-II.

Example

```
*EXIT<cr>
```

LOAD—Load 8086 Object Code

The Load command loads 8086 object code from the specified disk directory and file into 8086 memory. The Load command is the only way to load symbols and line numbers into the DEBUG-86 symbol table.

Command Syntax

```
LOAD [:Fn:]filename [{NOSYMBOL|NOLINE}...] <cr>
```

where

filename is the complete name of a disk file that is a valid absolute, PIC or LTL 8086 object module. A default extension is not assumed; you must provide an extension.

NOSYMBOL is a modifier that prevents the program symbol table from being loaded.

NOLINE is a modifier that prevents the program line number table (in PL/M-86 or PASCAL-86 programs) from being loaded.

Abbreviation

LOAD can be abbreviated to LOA, NOSYMBOL to NOS, and NOLOAD to NOL.

Description

The Load command allows you to load the local symbols and their types if present, source statement numbers, module names, segment and group names, and object code from the specified file.

For absolute programs, no checking is done for RAM being present. If your program is located in the memory address space where no memory exists, no warnings will be given.

For PIC and LTL code, the loader allocates only existing memory. The system prevents code from piling up in 8086 memory beyond previously loaded PIC/LTL code by deallocating 8086 memory space before loading. The system inserts symbolic information in the associated module and symbol tables in the order encountered. A base and displacement value are loaded for all symbols and statement numbers.

You can enter the NOSYMBOL and NOLINE modifiers in any order or combination. When you use more than one modifier, separate them with spaces. A modifier may not be specified twice in the same load command.

Possible Error Conditions

Invalid object files cause an error message to be displayed and abort the load operation.

Unsatisfied externals cause a warning to be displayed but do not abort the load operation.

Examples

1. To load an absolute, PIC, or LTL file from a disk in drive 0:

   ```
   *LOAD :F0:TEST.OBJ<cr>
   ```

2. To load an absolute, PIC, or LTL file from a disk in drive 2 without loading program symbols:

   ```
   *LOAD :F2:ABSOCODE2 NOSYMBOL<cr>
   ```

3. To load an absolute, PIC, or LTL file from a disk in drive 1 without loading the program statement numbers:

   ```
   *LOAD :F1:COUNTONE NOLINE<cr>
   ```

Execution Commands

This section describes the following commands:

GO	Executes your program until breakpoint conditions are met.
GR	Displays or changes the contents of GR.
STEP	Executes a single program instruction.

These commands allow you to specify the address where execution is to begin, and to specify the conditions for halting and returning control to the console for further commands.

After your program code is loaded, DEBUG-86 initializes for execution by loading the instruction pointer (IP) and code segment register (CS) with the address specified by the loaded object module.

In PIC and LTL code, SS and SP, and DS or ES may also be present with CS:IP.

GO—Execute 8086 Instructions

The GO command transfers control of the system to your program at the address specified or implied and executes instructions until breakpoint conditions, if any, are satisfied.

Command Syntax

```
GO [FROM address] { [FOREVER]                                          } <cr>
                  { [TILL break-address [OR break-address]]            }
                  { [TILL break-register [OR break-register]]          }
```

where

FROM *address* specifies the address of the first instruction to be executed. If FROM *address* is omitted, execution begins at the address in the IP and CS. The *address* must specify the CS:IP content in the form nnnn:nnnn, as in 800:0 (leading zeros need not be entered).

NOTE

If *address* is entered in the form 8000, for example, a relative jump could add a displacement to the value in the IP and cause an overflow. The CS:IP then would not reflect the desired value.

FOREVER is a function keyword that disables all breakpoint conditions.

TILL is a keyword introducing one or more breakpoint conditions.

break-address is an integer expression entered as a pointer that references a 20-bit execution address.

break-register is one of the breakpoint registers, BR0 or BR1. The address for BR forms a 20-bit memory address where DEBUG-86 writes a one-byte interrupt to get control.

Abbreviation

GO can be abbreviated to G, FROM to F, and TILL to T.

Description

The GO command begins real-time execution, optionally loading the CS and IP with a starting address, and continues until the breakpoint conditions are satisfied.

FOREVER Condition. When you enter a simple GO FOREVER command, execution begins at the current CS:IP address and continues until one of the following occurs:

- You enter a CNTL-D character. (If your program disables 8086 interrupts, the abort is disabled.)
- A fatal error occurs (explained in Chapter 7).
- Your program executes a system call to the EXIT routine.

When you enter the CNTL-D character, the following operations occur:

- DEBUG-86 completes executing the current instruction or system call.
- Execution halts; the IP and CS contain the address of the next instruction to be executed.
- The next instruction to be executed is disassembled.
- The message PROCESSING ABORTED is displayed, acknowledging the user abort.

Instead of having your program execute forever, you may specify one or two breakpoints.

Breakpoints. A breakpoint is the address of a program instruction that results in the return of control to DEBUG-86. A breakpoint allows you to check the contents of registers or data fields at the specified point in your program.

DEBUG-86 modifies user memory when entering execution and when a breakpoint is reached and repairs it when you return control to your program with the GO command. The instruction at the breakpoint address is not executed until your program regains control.

If a breakpoint is set at the starting address for resuming execution, a single step is executed, then the GO command, to enable you to keep a breakpoint the same when executing loops.

Breakpoint Registers. This option uses the contents of the breakpoint registers, BR0 and BR1, as breakpoint addresses.

To change the contents of either breakpoint register, or BR when setting both registers to the same value, use the form:

```
*break-register = address<cr>
```

To display the contents of either the BR0 or BR1 breakpoint register, use the form:

```
* break-register<cr>
```

Possible Error Conditions

Since DEBUG-86 must modify memory to get control, it cannot perform the break function in ROM. DEBUG-86 cannot enter GO with the 8086 Trap Flag (TFL) set.

Examples

1. *GO<cr>
2. *GO FROM 780:2<cr>
3. *GO FROM 780:2 TILL 780:9 OR 780:F<cr>
4. *GO FROM .+MOD 60<cr>
5. *G SF START+1 BR0<cr>
6. *GO FOREVER<cr>

GR Change/Display Go Register

The Go Register (GR) command changes or displays the contents of the go register.

Command Syntax

Display form:

```
GR<cr>
```

Change form:

```
GR = { FOREVER                                          } <cr>
     { TILL break-address [OR break-address]            }
     { TILL break-register [OR break-register]          }
```

where

FOREVER is a keyword that disables all breakpoint conditions. Execution stops only when you abort processing.

TILL is a keyword that introduces one or more breakpoint conditions.

break-address is an integer expression entered as a pointer that references a 20-bit execution address. The first *break-address* sets the contents of BR0; the second *break-address* sets the contents of BR1.

break-register is one of the breakpoint registers, BR0 or BR1 (or BR to denote both breakpoint registers), that is to be enabled.

Abbreviation

TILL can be abbreviated to T.

Description

The GR defines breakpoints conditions currently in force. By setting the contents of GR to a breakpoint condition (using the change form of the command), you do not have to re-enter the breakpoint conditions each time you restart execution with the GO command.

Entering GR<cr> displays the contents of GR.

Possible Error Conditions

Error 126, Symbol does not exist.
Error 137, Module does not exist.

Examples

1. *GR<cr>
   *
2. *GR=FOREVER<cr>
   *
3. *GR=TILL BR1<cr>
   *
4. *GR=TILL BR0 OR BR1<cr>
   *
5. *GR=AT 780:9 OR 780:F<cr>
   *
6. *GR=AT 780:9 OR 780:F<cr>

STEP—Execute a Single Instruction

The STEP command causes the execution of a single instruction in your program.

Command Syntax

```
STEP [FROM address]<cr>
```

where

FROM *address* specifies the address where single step execution is to begin. If FROM *address* is omitted, the address in the IP and CS is used. The *address* must specify the CS:IP contents in the form nnnn:nnnn, as in 800:0 (leading zeros need not be entered).

NOTE

IF *address* is entered in the form 8000, for example, a relative jump could add a displacement to the value in the IP and cause an overflow. The CS:IP then would not reflect the desired value.

Abbreviation

STEP can be abbreviated to S or STE, and FROM to F or FRO.

Description

When the STEP command is first issued, it initializes the user environment for single-step execution, optionally loading a starting address, and executes one instruction step. A subsequent step is executed each time you enter the STEP (or S) command.

After each step, the system displays the disassembled instruction to be executed next. If the step command is embedded in a compound command (described later in this chapter) the next instruction to be executed is not disassembled. This gives you the ability to make multiple steps and lets you decide whether to disassemble or not with the ASM form of the Display Memory command.

The Step command is a powerful debugging feature for repeat loops (compound commands) where you can give the terminating condition (UNTIL or WHILE) and display system status and values after each step.

Possible Error Conditions

Error 126, Symbols does not exist.
Error 137, Module does not exist.

Examples

1. *STEP<cr>
   *
2. *STEP FROM 80:2<cr>
   *
3. *S[illegible]MOD 60<cr>
   *
4. *S[illegible]0<cr>
   *
5. *S FROM CS:(WORD PTR 0) SHORT JUMP INDIRECT THROUGH 2<cr>
   *
6. *S[illegible]0<cr>

Change Commands

This section describes the Change commands that allow you to change the contents of 8086 registers, I/O ports, and memory. The Change commands are:

Change Register	Changes the contents of a single register or status flag.
Change Memory	Changes the contents of 8086 memory locations.
Change Port	Changes the contents of I/O ports.

Change Register—Change Content of a Register

The Change Register command changes the contents of a single 8086 processor register or status flag.

Command Syntax

register = change-exp<cr>

where

register is one of the following keyword references:

8086 Register Type	Keyword References
8-bit registers	RAL, RAH, RBL, RBH, RCL, RCH, RDL, RDH
16-bit registers	RAX, RBX, RCX, RDX, SP, BP, SI, DI, SS, CS, DS, ES, IP, RF
1-bit status flags	CFL, PFL, AFL, ZFL, SFL, TFL, IFL, DFL, OFL

change-exp is a numeric expression specifying the new contents of *register*.

Abbreviation

None

Description

If *change-exp* contains fewer bits than the specified register, the bits are right-justified and the unspecified high-order bits are set to zeroes.

If *change-exp* contains more bits than the specified register, the extra high-order bits are lost.

Possible Error Conditions

Error 148, Integer value required
Error 147, Pointer value required

Examples

1. To change the contents of AX to 0000H:

```
*RAX=0<cr>
*
```

2. To change the contents of IP to F23AH:

```
*IP=#F23A<cr>
*
```

3. To change the contents of CS to the word value located at WORD.SAM:

```
*CS=#WORD.SAM<cr>
*
```

Change Memory—Change Contents of Memory Locations

The Change Memory command changes the contents of one or more memory locations.

Command Syntax

memory-type address {[TO *end-address*] / [LENGTH n]} = *change-exp* [,...]19<cr>

where

memory-type is one of the following keywords: BYTE, WORD, SINTEGER, INTEGER, POINTER.

address is a memory location entered as a pointer value containing a base (which can be omitted if 0) and a displacement. If a range is accessed, *address* is the starting address. *address* can be either:

- A numeric expression, the result of which is an address modulo 65536.
- A memory content reference of the form (*memory-type address*), as in BYTE(WORD 1000). The content of the address or address pair inside the parentheses is treated as the address for the memory-type outside the parentheses.

TO *end-address* specifies the upper limit of a range of memory that is to be modified. The *end-address* must be greater than or equal to *address*. Both addresses must have the same base.

LENGTH n specifies the number of bytes, words, or pointers (depending on *memory-type*) to be modified. The value n must be an integer value.

change-exp is the value to replace the contents of the specified memory location. Up to 19 *change-exp*s may be listed. The *change-exp* must be a pointer value if *memory-type* is a pointer; otherwise, it must be an integer value. The *change-exp* may be any of the following:

- A numeric expression, representing a single value.
- A string, enclosed in quotes, that represents consecutive bytes consisting of the ASCII equivalents of the string characters.
- A range of memory addresses, representing consecutive bytes, words, or double words of user memory.

Abbreviation

BYTE can be abbreviated to BYT, WORD to WOR, SINTEGER to SIN, INTEGER to INT, POINTER to POI, and LENGTH to LEN

Description

To change the contents of a single memory location, use the form:

memory-type address = *change-exp*<cr>

To change the contents of a range of memory locations, use any of the following forms:

1. *memory-type* address = *change-exp* [,...]19<cr>

   In this form the upper limit of the destination range is implicit in the number of *change-exp*s listed.

2. *memory-type* address TO *end-address* =
   change-exp [,...]19<cr>

   If the number of *change-exps* is smaller than the range, the system repeats values as necessary. If larger than the range, the extra values are lost and an error occurs.

   With this form, you access memory depending on *memory-type* specified, as follows:

   - BYTE or SINTEGER. You access each byte location in the range, including *address* and *end-address*.
   - WORD or INTEGER. Address pairs are accessed until *end-address* is reached. If *end-address* is the low address of a pair, the last word is formed from *end-address* and the next address.
   - POINTER. Address quadruples are accessed until *end-address* is reached. If *end- address* is the low address of a quadruple, the last pointer is formed from *end-address* and the next three addresses; if *end-address* is not the low address of a quadruple, the access ends after the previous quadruple.

3. *memory-type* address LENGTH n = *change-exp* [,...]19<cr>

   This form specifies the actual number of bytes, words, or pointers to be modified. Access can begin with even- or odd-numbered address. The range is filled as described under the second form above.

Possible Error Conditions

Error 149, Invalid base error
Error 124, Partition bounds error
Error 148, Integer value required
Error 127, Memory failure
Error 139, Excessive data

Examples

1. To change the byte contents of a single memory location 800:30H

   ```
   *BYTE 800:30 = FF<cr>
   *
   ```

2. To change the byte contents of a range of memory locations to a single value:

   ```
   *BYTE 800:30 LEN 16 = 00H<cr>
   *
   ```

3. To replace the byte contents of a range of memory locations with a list of new values:

   ```
   *BYT 800:30 TO 800:FF = 12, 34, 56, 78, 9A, BC<cr>
   *
   ```

4. To replace the word contents of a single memory location with the contents of the register IP:

   ```
   *WORD 800:30 = IP<cr>
   *
   ```

5. To increment the contents of a single memory location:

   ```
   *WOR 800:30 = (WOR 708:3) + 1<cr>
   *
   ```

6. To change the pointer contents of a single memory location:

   ```
   *POINTER 800:30 = ABCD:1234<cr>
   *
   ```

7. To replace the byte contents of a range of memory with a string:

```
*BYTE 300:30 = "ABCDE"<cr>
*
```

8. To replace the pointer contents of a range of memory with the contents of a second range of memory:

```
*POI 300:30 = POI 300:51 LEN 20<cr>
*
```

9. To change the byte contents of a memory location referenced by a statement number:

```
*BYT #56 = "FA<cr>
*
```

Change Port—Change Contents of I/O Ports

The Change Port command changes the contents of one or more I/O ports.

Command Syntax

port-type address { [TO *end-address*] / [LENGTH n] } = *change-exp* [,...]19<cr>

where

port-type is one of the following:

- PORT—references the 8-bit port value at *address*.
- WPORT—references the 16-bit port value at *address* and *address* + 1, one byte at a time and not as a single 16-bit port value.

address is the address of an 8086 port and is an integer value between 0 through 65,535 inclusive. If a range of ports is specified, *address* is the starting address of the range.

TO *end-address* specifies the upper limit of a range of port addresses. The *end-address* is an integer value between 0 and 65,535 inclusive, and must be greater than or equal to *address*. Both addresses must have a base of zero.

LENGTH n specifies the number of port or word port addresses to be displayed. The value of n must be an integer.

change-exp is the value to replace the contents of the specified port. Up to 19 *change-exps* of the following types may be listed:

- A numeric expression that represents a single value.
- A string enclosed in quotes that represents consecutive bytes consisting of the ASCII equivalents of the string characters.
- A range of memory or port addresses, representing consecutive bytes or words of user memory or I/O ports.

Abbreviation

PORT can be abbreviated to POR, WPORT to WPO, and LENGTH to LEN.

Description

To change the contents of a single port, use the form:

port-type address = change-exp<cr>

To change the contents of a range of ports, use one of the following forms:

1. *port-type address = change-exp* [,...]19<cr>
   The upper limit of the destination range is implicit in the number of *change-exps* listed.

2. *port-type address* TO *end-address = change-exp* [,...]19<cr>
   If the number of *change-exps* is smaller than the range, values are repeated as necessary to fill the range. If the number is larger than the range, the excess values are lost and an error occurs.

3. *port-type address* LENGTH n = *change-exp* [,...]19<cr>
   This form specifies the actual number of ports to be modified. Access can begin on an even- or odd-numbered address. The specified range is filled as described under No. 2 above.

Possible Error Conditions

Error 148, Integer value required
Error 124, Partition bounds error
Error 139, Excessive data

Examples

1. To change the contents of a single byte port:
   *PORT 08 = #41<cr>
   *
2. To change the contents of a range of byte ports:
   *PORT 30 = #15, #22, #33, #44, #55, #66<cr>
   *
3. To replace the contents of a range of ports with a string:
   *PORT 1000 TO 1005 = "BCDE"<cr>
   *
4. To fill a range of ports with the same value:
   *POR 1000 LEN 5 = FF<cr>
   *
5. To attempt to fill a range with too many values:
   *POR 1000 TO 1002 = #5, #22, #33, ..#3<cr>
   *
   An error message such as ERROR 140 is displayed.

Display Commands

This section describes the commands that enable you to display the following systems elements:

- 8086 processor registers
- 8086 status registers
- Memory
- Ports
- Status flags

The Display commands are:

Display Registers	Displays contents of 8086 registers.
Display Memory	Displays contents of 8086 memory locations.
Display Memory (ASM Form)	Displays contents of 8086 memory locations in 8086 Assembly language mnemonics.
Display Port	Displays contents of I/O ports.
Display Stack	Displays contents of user's stack.
Display Boolean	Displays boolean value of input.
Evaluate	Displays an integer value in five number bases or a pointer value in five hexadecimal digits.

Display Register—Display Contents of 8086 Registers

The Display Register command displays the contents of one or more 8086 processor registers and status flags.

Command Syntax

```
{ register[, ...]19 }
{    REGISTER       } <cr>
{     FLAG          }
```

where

register is any of the following keyword references (up to 19 can be entered, separated by spaces):

8086 Register Type	Keyword References
8-bit registers	RAL, RAH, RBL, RBH, RCL, RCH, RDL, RDH
16-bit registers	RAX, RBX, RCX, RDX, SP, BP, SI, DI, SS, CS, DS, ES, IP, RF
1-bit status flags	CFL, PFL, AFL, ZFL, SFL, TFL, IFL, DFL, OFL

REGISTER is a command keyword that causes the display of all the 16-bit 8086 registers.

FLAG is a command keyword that displays all the 1-bit status flags.

Abbreviation

REGISTER can be abbreviated to R or REG, and FLAG to FLA.

Description

The referenced values are displayed on 80-column-wide lines separated by spaces according to the appropriate format, as follows:

8-bit-register-name = byte
16-bit-register-name = word
status-flag-name = bit

Possible Error Conditions

Syntatic errors are possible.

Examples

1 To display the AX, BH, SP and AFL registers:

```
*RAX,RBH,SP,AFL<cr>
RAX=0001H RBH=2FH SP=FFE7H AFL=0<cr>
```

2. To display all nineteen 16-bit registers:

```
*REGISTER<cr> or *R<cr>
RAX=000H RBX=00A2H RCX=0001H RDX=0010H SP=000AH BP=0000H SI=0123H DI=0000H
CS=0000H DS=FF1EH SS=0000H ES=0000H RF=0000H IP=FABCH
```

3. To display the nine status flags:

```
*FLA<cr>
CFL=0 PFL=0 AFL=0 ZFL=0 SFL=0 TFL=1 IFL=0 DFL=0 OFL=0
```

Display Memory—Display 8086 Memory

The Display Memory command displays the contents of one or more 8086 memory locations.

Command Syntax

memory-type address { [TO *end-address*] / [LENGTH n] } <cr>

where

memory-type is one of the following keywords: BYTE, WORD, SINTEGER, INTERGER, POINTER.

address is a pointer value that contains a base (which need not be entered if 0) and a displacement and specifies an address of a memory location. If a range is specified, *address* is the starting address in the range. The *address* can be either:

- A numeric expression, the result of which becomes an address modulo 65536.
- A memory content reference in the form (*memory-type address*), as in BYTE(WORD 1000). The content of the address or address pair inside the parentheses is treated as the address for the memory-type outside the parentheses.

TO *end-address* specifies the upper limit of a range of memory. The *end-address* must be greater than or equal to *address*. Both addresses must have the same base.

LENGTH *n* specifies the number of bytes, words, or pointers to be displayed. The value *n* must be an integer.

Abbreviation

BYTE can be abbreviated to BYT, WORD to WOR, SINTEGER to SIN, INTEGER to INT, POINTER to POI, and LENGTH to LEN

Description

To display the contents of a single memory location, use the form:

memory-type address<cr>

To display the contents of a range of memory locations use either of two forms:

1. *memory-type address* TO *end-address*<cr>
2. *memory-type address* LENGTH n<cr>

In 1 and 2 above, memory is accessed as described under the Change Memory Command.

Each line of display contains the memory address of the first value followed by a maximum number of values depending on memory type, as follows: BYTE—16, WORD—8, SINTEGER—8, INTEGER—8, POINTER—4.

Possible Error Conditions

Error 149, Differing bases
Error 124, Partition bounds error

Examples

1. To display the byte contents of location 800:30H:

```
*BYTE 800:30<cr>
BYT 0800:0030H=66H
```

2. To display the byte contents of locations 800:30H through 800:30H:

```
*BYT 800:30 LEN 10<cr>
BYT 0800:0030H=66H 6FH 72H 20H 70H 72H 6FH 67H 72H 61H
```

3. To display the word contents of location 800:30H:

```
*WORD 800:30 <cr>
WOR 0800:0030H = EF06H
```

4. To display 16H words beginning at location 800:30H:

```
*WOR 800:30 LEN 16<cr>
WOR 0800:0030H=6F66H 2072H 7270H 676FH 6172H 736DH
```

5. To display the pointer contents of location 800:30H:

```
*POINTER 800:30<cr>
POI 0800:0030H=2072:6F66H
```

6. To display the POINTER contents of locations 800:30H through 800:3BH:

```
*POI 800:30 TO 800:3B<cr>
POI 0800:0030H=2072:6F66H 676F:7270 736D:6172H
```

Display Memory (ASM form)—Display 8086 Memory in ASM Form

The ASM form of the Display Memory command displays the contents of one or more 8086 memory locations in 8086 Assembly language mnemonics.

Command Syntax

```
ASM address {[TO end-address]} <cr>
            {[LENGTH n]       }
```

where

address is a pointer value that contains a base and a displacement and specifies an address of a memory location. If a range is specified, *address* is the starting address in the range. The *address* can be either:

- A numeric expression, the result of which becomes an address modulo 65536.
- A memory content reference in the form (*memory-type address*), as in BYTE(WORD 1000). The content of the address or address pair inside the parentheses is treated as the address for the memory-type outside the parentheses.

TO *end-address* specifies the upper limit of a range of memory. *end-address* must be greater than or equal to *address*. Both addresses must have the same base.

LENGTH n specifies the number of bytes, words, or pointers to be displayed. The value of n is an integer.

Abbreviation

LENGTH can be abbreviated to LEN

Description

The specified range of memory is disassembled into 8086 Assembly language in the following format:

```
Address          Prefix   Mnemonic   Operands   Comments

FF00:0390                   IN        AL, DX
FF00:0391                   TEST      AL,07H
FF00:0393                   JE        $-06H      ;SHORT
```

4. To display 16H words beginning at location 800:30H:

```
*WOR 800:30 LEN 16<cr>
WOR 0800:0030H=6F66H 2072H 7270H 676FH 6172H 736DH
```

5. To display the pointer contents of location 800:30H:

```
*POINTER 800:30<cr>
POI 0800:0030H=2072:6F66H
```

6. To display the POINTER contents of locations 800:30H through 800:3BH:

```
*POI 800:30 TO 800:3B<cr>
POI 0800:0030H=2072:6F66H 676F:7270 736D:6172H
```

Display Memory (ASM form)—Display 8086 Memory in ASM Form

The ASM form of the Display Memory command displays the contents of one or more 8086 memory locations in 8086 Assembly language mnemonics.

Command Syntax

```
ASM address {[TO end-address]} <cr>
            {[LENGTH n]       }
```

where

address is a pointer value that contains a base and a displacement and specifies an address of a memory location. If a range is specified, *address* is the starting address in the range. The *address* can be either:

- A numeric expression, the result of which becomes an address modulo 65536.
- A memory content reference in the form (*memory-type address*), as in BYTE(WORD 1000). The content of the address or address pair inside the parentheses is treated as the address for the memory-type outside the parentheses.

TO *end-address* specifies the upper limit of a range of memory. *end-address* must be greater than or equal to *address*. Both addresses must have the same base.

LENGTH n specifies the number of bytes, words, or pointers to be displayed. The value of n is an integer.

Abbreviation

LENGTH can be abbreviated to LEN

Description

The specified range of memory is disassembled into 8086 Assembly language in the following format:

```
Address         Prefix   Mnemonic   Operands   Comments

FF00:0390                  IN        AL, DX
FF00:0391                  TEST      AL,07H
FF00:0393                  JE        $-06H      ;SHORT
```

where

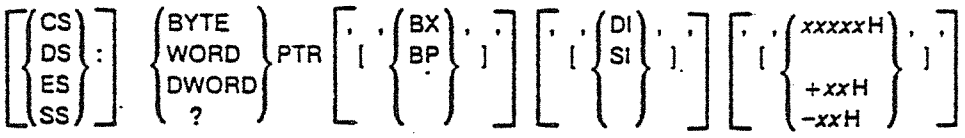
ADDR marks the first byte (or prefix) of the instruction in hexadecimal.

PREFIX identifies any prefix byte other than segment override (such as LOCK, REPE, REPNE) that may be specified. If none is specified, the column is blank.

MNEMONIC is the 8086/8087/8088 macro assembler mnemonic for the instruction.

OPERANDS define zero or more operands separated by commas. Register operands are any of the 8086 registers: AL, AH, BL, BH, CL, CH, DL, DH, AX, BX, CX, DX, SI, DI, BP, SP.

The disassembled memory operands used by DEBUG-86 have the following format:



Example: the display ES:BYTE PTR [BX] [SI] [+01H]
represents the operand BYTE ES:(BX+SI+1)

where

- The first field is the segment register field and is only displayed if the instruction has a segment-override prefix.
- In the second field, an entry "?PTR" means that the type of the pointer cannot be determined from the context.

  Example: `LEA AX,? PTR [34A0H]`
- In the third and fourth fields the base register (BX, BP, and index register (DI, SI) fields are not displayed for direct memory operands. When these fields are displayed, they are enclosed in brackets.
- The last field is either a 16-bit unsigned (word) number, or a signed 8-bit (byte) number. The entry is displayed enclosed in brackets.
- At least one of the last three fields (base register, index register, number) is displayed for any memory operand.

Examples

1. To display the contents of address 800:9C7H in ASM form:

```
*ASM 800:9C7<CR>
ADDR        PREFIX  MNEMONIC  OPERANDS                COMMENTS
0800:09C7H          MOV       AX, WORD PTR [1848H]
```

2. To display the contents of a range of addresses in ASM form:

```
*ASM 800:0BBC LEN 5 <CR>
ADDR        PREFIX  MNEMONIC  OPERANDS                COMMENTS
0800:0BBCH          SBB       BYTE PTR [BX][SI],AL
0800:0BBEH          JE        $+39H                   ;SHORT
0800:0BC0H          CALL      $+003BH                 ;SHORT
```

Display Port—Display I/O Port Contents

The Display Port command displays the contents of one or more I/O ports.

Command Syntax

port-type address {[TO *end-address*] / [LENGTH n]} <cr>

where

port-type is one of the following keywords:

- PORT—references the 8-bit port value at *address*.
- WPORT—references the 16-bit port value at *address* and *address* + 1, one byte at a time and not as a single 16-bit port value.

address is the address of an 8086 port and is an integer value between 0 through 65,535 inclusive. If a range is specified, *address* is the starting address of the range.

TO *end-address* specifies the upper limit of a range of port addresses. *end-address* is an integer value between 0 and 65,535 inclusive, and must be greater than or equal to *address*.

LENGTH n specifies the number of port or word port addresses to be displayed. The value n must be an integer.

Abbreviation

PORT can be abbreviated to POR, WPORT to WPO, and LENGTH to LEN.

Description

To display the contents of a single port, use the form:

port-type address<cr>

To display the contents of a range of ports, use either the TO *end-address* form or the LENGTH n form.

Possible Error Conditions

Error 148, Integer value required
Error 124, Partition bounds error

Examples

1. To display the contents of the byte port at address 120H:

```
*PORT 120<cr>
POR 0120H=BFH
```

2. To display the contents of a range of byte ports:

```
*PORT 120 TO 126<cr>
POR 0120H=BFH 7FH BFH 7FH BFH 7FH
```

3. To display the contents of a single word port at address 120H:

```
*WPORT 120<cr>
WPO 0120H=7FBFH
```

4. To display the contents of a range of word ports:

```
*WPORT 120 TO 128<cr>
WPO 0120H=7FBFH 7FBFH 7FBFH 7BBFH 7BBFH
```

Display Boolean—Display Boolean Value

The Display Boolean command displays the boolean value of the input value.

Command Syntax

BOOL *expression*<cr>

where

expression is an integer expression, the result of which is evaluated to a boolean value. If the least significant bit of the result equals 1, the boolean value is TRUE; otherwise the boolean value is FALSE.

Abbreviation

BOOL can be abbreviated to BOO.

Description

A boolean expression is an expression that is contained in a boolean command context. The least significant bit of the result is tested to obtain a TRUE or FALSE value. Any integer value may be used in a boolean context.

A boolean expression uses relational and logical operators to manipulate TRUE/FALSE values. When a relational operator is evaluated, the result is always either 0 (FALSE) or FFFFH (TRUE).

Possible Error Conditions

Error 148, Integer value required

Examples

1. To display the boolean value of FFH:

```
*BOOL 0FFH<cr>
TRUE
*
```

2. To display the boolean vaue of an expression that includes logical and relational operators:

```
*BOOL CS:DS AND (P > 60)<cr>
FALSE
*
```

3. To display the boolean value of an expression that includes a symbol:

```
*BOO BYTE <X> F2<CR>
FALSE
*
```

Display Stack—Display User Stack Contents

The Display Stack command displays the contents of the user's stack.

Command Syntax

```
STACK expression<cr>
```

where

expression is an integer expression, the value of which defines the number of words on the user stack to be displayed.

Abbreviation

STACK can be abbreviated to STA.

Description

The stack is located in user memory referenced by the pointer value SS:SP. The specified number of words are displayed from the top of the stack.

Possible Error Conditions

Error 148, Integer value required
Error 126, Symbol does not eixst

Examples

1. To display the contents of five words at the top of the stack:

```
*STACK 5<CR>
WOR 0839:0DCCH=4100H 4342H 2045H 3D30H 6874H
```

2. To display the contents of n stack words, where n is the word value stored at .SAM:

```
*STA .SAM<CR>
```

EVALUATE—Display Integers in Five Bases

The Evaluate command displays integer values in binary, octal, decimal, hexadecimal, and ASCII, and pointer values in pointer and 20-bit forms.

Command Syntax

```
EVALUATE expression [SYMBOLICALLY]<cr>
```

where

expression is an integer expression.

SYMBOLICALLY is a keyword that displays each numeric value output by the command as as a symbol or a source statement, plus a remainder. The numeric value is assumed to be a pointer. If no symbol with a value less than the value being evaluated exists, it is displayed in pointer form.

Abbreviation

EVALUATE can be abbreviated to EVA, and SYMBOLICALLY to SYM.

Description

The Evaluate command translates integers from one base to another and computes a 20-bit address like that of a pointer. Any expression is evaluated to a single number and the result is displayed in binary, octal, decimal, hexadecimal, and ASCII bases.

In the four numeric bases, the BYTE and WORD values have a suffix and sufficient leading zeroes to contain the following number of digits.

	Hexadecimal (H)	Decimal (T)	Octal (Q)	Binary (Y)
BYTE	2	3	3	8
WORD	4	5	6	16

In base ASCII, characters are enclosed in single quotes ('); printing characters (ASCII codes 20H through 7EH after bit 7 is masked off) are displayed, while non-printing characters are suppressed.

If you specify the keyword SYMBOLICALLY, the system searches the symbol table for the symbol or statement number with the same base whose offset value is closest to but not greater than the value being output. If a symbol and a statement number have the same value, the symbol is used. The value is then displayed as either (symbol + numeric constant) or (statement number + numeric constant), where the numeric constant is the non-zero remainder in hexadecimal base. If no symbol or statement number has a value less than or equal to the value, it is output as a numeric constant.

Possible Error Conditions

Error 126, Symbol or line does not exist
Error 137, Module does not exist

Examples

1. To display the value of 4142H in the five bases:

```
*EVA 4142<cr>
1000001010000Y 40502Q 16706T 4142H 'AB'
*
```

2. To display the value of FFH + ADH in five bases:

```
*EVA FF+AD<cr>
110101100Y 654Q 428T 1ACH ','
*
```

3. To display 111:0222H as a symbol or statement number plus remainder:

```
*EVA 111:0222 SYM<cr>
0111:0222H
*
```

This example assumes that no symbol or statement number match.

4. To display 111:222H as a symbol or statement number plus remainder:

```
*E'A '111:222<cr>
..MOD1.SAM + 0021H
*
```

This example assumes that a matching symbol with an address at 111:201 has been selected.

Symbol Manipulation Commands

The commands described in this section allow you to manipulate the symbols and statement numbers in the DEBUG-86 symbol and statement number tables. With these commands, you create and change symbols, and display and remove symbols, modules, and source statement numbers.

The Symbol Manipulation commands are:

DEFINE Symbol	Enters a new symbol in the table.
Display Symbols	Displays symbols and their values.
Display Lines	Displays statement numbers and associated absolute addresses.
Display Modules	Displays the names of all modules.
Change Symbols	Changes the value and type of symbols.
Remove Symbols	Removes specified symbols, or specified modules, or all modules, symbols, and statement numbers.
Set Domain	Establishes a default module for statement number references that do not contain a module name.

Define Symbol—Enter New Symbol

The Define Symbol command enters new symbols in the DEBUG-86 symbol table.

Command Syntax

```
DEFINE [..module].symbol = change-exp [OF memory-type]<cr>
```

where

module is the name of an existing program module, in which *symbol* is to be located. The *module* is prefixed by two periods (..).

symbol is a user-defined symbol to be entered into the symbol table for use during the debugging session. The *symbol* references a location in the symbol table, and is prefixed by a period (.).

change-exp is a numeric expression, the value of which is to be assigned to *symbol*. The *change-exp* represents an address of statement labels or variables, or the value of a constant.

OF *memory-type* specifies any of the following: BYTE, WORD, SINTEGER, INTEGER, or POINTER. If *memory-type* is omitted, *symbol* has no type.

Abbreviation

DEFINE can be abbreviated to DEF.

Description

The rules for defining new symbols are as follows:

You can use up to 122 characters for the symbol name, of which the first 31 characters must be a unique combination.

The first character in the symbol name must be an alphabetic character, or one of the two characters @ or ?. The remaining characters can be these characters or numeric digits.

You can specify the module that is to contain the new symbol. The module named must already exist in the table. The symbol is then placed in that module's section of the table. Symbols defined without a module are placed in the unnamed module at the head of the table.

The new symbol name may not duplicate a symbol name already present in the named module. However, the same symbol name may appear in different modules.

Possible Error Conditions

Error 125, Symbol already exists
Error 137, Module does not exist

Examples

1. To enter the byte symbol .BEGIN in the symbol table module ..MAIN with a value of F3H:

   *DEFINE ..MAIN.BEGIN = F3 OF BYTE<cr>

   *

2. To enter the untyped symbol .CAR into the symbol table with a value of 0000:0F00H:

   *DEF .CAR = 0000:0F00<cr>

   *

3. To enter the word symbol ENT1 into the symbol table with a value of .VAR + 10:

   *DEF .ENT1 = .VAR + 10 OF WORD<cr>

   *

4. To enter a pointer symbol .CAT2 into the symbol table with a value of 0700:0050H:

   *DEF .CAT2 = 0700:0050 OF POI<cr>

   *

5. To enter a pointer symbol .CAT2 in the module ..SUBRA with a value of 0000:00F0H:

   *DEF ..SUBRA.CAT2 = 0000:00F0 OF POI<cr>

   *

Display Symbols—Display One or More Symbols

The Display Symbols command displays one or more symbols (and modules if any).

Command Syntax

```
{          SYMBOL                              } <cr>
{[..module].symbol [.symbol] ...}
```

where

SYMBOL is a command keyword that causes the entire DEBUG-86 symbol table to be displayed, module by module.

module is the name of the program module in which *symbol* is located. The *module* is prefixed by two periods (..).

symbol is the name of a symbol that references a location in the symbol table. The *symbol* is prefixed by a period (.).

Abbreviation

SYMBOL can be abbreviated to SYM.

Description

To display a value from the symbol table, enter the symbol name (and module name if any). If the symbol desired is the first occurrence of that symbol in the symbol table, you do not need the module name. The symbol table value is displayed on the next line.

To display the entire symbol table, enter the command SYMBOL. Symbols are displayed module by module, starting with the unnamed module. Each module name is displayed at the head of that module's symbols. The value corresponding to each symbol is also displayed.

Possible Error Conditions

Error 137, Module does not exist
Error 126, Symbol does not exist

Examples

1. To display the symbol .SAM:

```
*.SAM<cr>
.SAM=0200:1FE2H OF INT
```

2. To display the symbol .SAM located in module ..MYPROG:

```
*..MYPROG.SAM<cr>
.SAM=0200:1FE2H OF INT
```

3. To display the entire symbol table:

```
*SYMBOL<cr>
.TEMP=0000.0001H
MODULE ..MAIN
.BEGIN=0800:0050H
```

```
.VAR=0800:0100H OF BYT
MODULE ..SUBR
.PROC=0800:0069H
.X=0800:0101H OF WOR
```

Display Lines—Display Statement Numbers

The Display Lines command displays the value of a single source statement number or all statement numbers.

Command Syntax

```
{LINE                              }
{[..module]#statement-number       } <cr>
```

where

LINE is a command keyword that displays all statement numbers and associated absolute addresses in the current domain. The module name is printed at the head of the module's line numbers when displaying all lines.

module is the name of the program module in which *statement-number* is located. It is prefixed by two periods.

statement-number is the source statement number. It is a numeric constant with a default suffix that is always decimal. It is prefixed by a number sign (#).

Abbreviation

LINE can be abbreviated to LIN.

Description

DEBUG-86 maintains a statement number table for PL/M-86 or PASCAL-86 program source codes. The statement numbers are assigned by the compiler. The address of the first instruction generated by each source statement corresponds to each source statement number in the table.

When you issue the command, LINE, each module name is printed at the head of the module's line numbers.

To display the value of a single source statement number and associated absolute address, specify the statement number prefixed by a number sign (#). If two or more modules have been compiled separately and contain the same statement numbers, specify the module name.

DEBUG-86 does not allow you to change the address corresponding to an existing statement number, or define any new statement numbers, or delete (remove) any statement numbers.

Possible Error Conditions

Error 137, Module does not exist
Error 126, Statement number does not exist

Examples

1. To display the value from the number table of a single statement number:

```
*#1<cr>
#1=0800:0050H
```

2. To display the value from the number table of a single statement number contained in a particular module:

```
*.MAIN#2<cr>
#2=0800:0057H
```

3. To display the addresses of all the statement numbers in the statement number table:

```
*LINE<cr>
MODULE ..MAIN
#1=0800:0050H
#2=0080:0057H
MODULE ..SUBR
#1=1140:0012H
#2=1140:0037H
#3=1140:00DFH
```

Display Modules—Display Module Names

The Display Modules command displays the names of all the modules currently in the DEBUG-86 symbol table.

Command Syntax

```
MODULE<cr>
```

Abbreviation

MODULE can be abbreviated to MOD

Description

To display the names of all the modules currently in the symbol table, enter the keyword MODULE.

Possible Error Conditions

None

Example

```
*MODULE<cr>
MODULE ..MAIN
MODULE ..SUBR
```

Change Symbols—Change Value of a Symbol

The Change Symbol command changes the value (and memory-type if specified) of a symbol.

Command Syntax

```
[..module].symbol[.symbol ...] ... = change-exp [OF memory-type]<cr>
```

where

module is the name of the program module in which *symbol* is located and is prefixed by two periods (..).

symbol is the name of an existing symbol that references a location in the symbol table. Each *symbol* is prefixed by a period (.).

change-exp is a numeric expression of a pointer value to be assigned to *symbol* and represents either the address of statement labels or variables, or the value of a constant.

OF *memory-type* specifies the memory type of *symbol*: BYTE, WORD, SINTEGER, INTEGER, or POINTER. If OF *memory-type* is omitted, the symbol's memory type is not changed.

Abbreviation

BYTE can be abbreviated to BYT, WORD to WOR, SINTEGER to SIN, INTEGER to INT, and POINTER to POI.

Description

The Change Symbol command replaces the value (and memory-type if specified) of a symbol.

When the same symbol name exists in different modules, specify the name of the module containing the desired symbol.

Possible Error Conditions

Error 137, Module does not exist
Error 126, Symbol does not exist

Examples

1. To change the value of a single symbol in the symbol table:

   ```
   *.ABC = 2000<cr>
   *
   ```

2. To change the value and memory type of a symbol located in a particular module:

   ```
   *..MAIN.DEF = 50 OF WORD<cr>
   *
   ```

3. To replace the value of a symbol with the sum of the values of two other symbols, and change the memory type:

   ```
   *.TEMP = .ABC + ..MAIN.DEF OF WORD<cr>
   *
   ```

Remove Symbols—Remove Symbols/Modules

The Remove Symbols command removes one or more symbols, or one or more modules, or all symbols and lines from the DEBUG-86 tables.

Command Syntax

```
        ⎧[..module].symbol [.symbol...]19 , ...⎫
REMOVE  ⎨SYMBOL                                ⎬  <cr>
        ⎩MODULE ..module [,..module]...        ⎭
```

where

module is the name of an existing program module in the symbol table, and is prefixed by two periods (..). Up to 19 modules can be listed at one time.

symbol is the name of an existing symbol in the symbol table, and is prefixed by a period (.). Up to 19 symbols can be listed at a time.

SYMBOL is a command modifier that deletes the entire current DEBUG-86 symbol table.

MODULE is a command modifier that deletes all the symbols and lines of the named module from the symbol and statement number tables. The object code is not affected.

Abbreviation

REMOVE can be abbreviated to REM, SYMBOL to SYM, and MODULE to MOD.

Description

To remove one or more symbols from the symbol table, use the first form of the command syntax. If the desired symbols occur in more than one module, specify the module name. The first occurrence of each symbol is deleted.

To remove all modules, symbols and statement numbers from both tables, use the second form of the command syntax.

To remove a single module, use the third form of the command syntax. Removing a module removes all symbols and statement line numbers lines in the module, but does not affect the object code.

When more than one module is listed, separate them with commas. When more than one symbol is listed, separate them with spaces.

Possible Error Conditions

Error 139, More than 20 symbols entered
Error 137, Module does not exist
Error 126, Symbol does not exist

Examples

1. To remove a single symbol from the symbol table:
   *REMOVE ABC<cr>
   *
2. To remove a symbol from a particular module:
   *REMOVE ..MAIN.DEF<cr>
   *
3. To remove symbols from different modules:
   *REM ..HIJ,..PARM1,..MAIN.TWO,..CARS,..CARS1,..SUBR.XX<cr>
   *
4. To delete both the symbol table and the statement number table entirely:
   *REMOVE SYMBOL<cr>
   *
5. To remove a single module from the symbol and statement number tables:
   *REMOVE MODULE ..MAIN<cr>
   *
6. To remove three modules from the symbol and statement number tables:
   *REM MOD ..MAIN,..SUBR,..CALC<cr>
   *

Set Domain—Establish Default Module

The Set Domain command establishes a specified module as the default module for statement number references.

Command Syntax

```
DOMAIN ..module<cr>
```

where

DOMAIN is a command keyword that establishes a default module for source statement number references.

module is the name of an existing program module in the statement number table and is prefixed by two periods (..).

Abbreviation

DOMAIN can be abbreviated to DOM.

Description

This command establishes a default module so that you need not specify the module name each time you reference a statement number contained in that module.

When the domain is set, you need not use module names on statement numbers while debugging that portion of the program.

Possible Error Conditions

Error 137, Module does not exist

Example

1. To establish the module ..MAIN as the default:

```
*DOMAIN=..MAIN<cr>
*
```

Compound Commands

The compound commands described in this section enhance the operation of DEBUG-86 by extending the power of the simple commands. A compound command is a control structure that contains zero or more commands.

The compound commands are:

REPEAT	A looping command
COUNT	A looping command
IF	A conditional execution command

The examples in this section are independent of each other. The introduction to each example gives the initial conditions for that example, and does not assume any results or conditions from any previous examples.

REPEAT Command

The REPEAT command executes zero or more DEBUG-86 commands in a loop; the loop can also contain zero or more logical conditions for termination.

The REPEAT command consists of the REPEAT keyword, zero or more commands of any type, zero or more exit conditions using WHILE or UNTIL, and the keyword END. Enter each of these elements on its own line of the console display. Terminate each input line with an intermediate carriage return (shown as *cr* in the command syntax). Terminate the last line, END, with a final carriage return to begin the sequence of execution.

Syntax

```
REPEAT<cr>
[command<cr>
 WHILE boolean-expression<cr>] ...
 UNTIL boolean-expression<cr>]
END<cr>
```

Description

After each intermediate carriage return, the system begins the next line with a period (giving an indented appearance), then the asterisk prompt to signal readiness to accept the next line. The END keyword can be entered as ENDR or ENDREPEAT; the characters after END serve as a form of "comment" to indicate which loop is being terminated.

The elements to be repeated are shown in brackets in the syntax. Each element can be a command, a WHILE clause, or an UNTIL clause. You can mix these elements in any order, using any number of each type of element.

Each command is executed when it is encountered on each iteration. After the command has been completely executed, the loop proceeds to the next element.

The WHILE and UNTIL keywords introduce exit clauses. The WHILE clause terminates execution of the loop when its boolean-expression evaluates FALSE. The UNTIL clause terminates the loop when its boolean-expression evaluates TRUE.

In both the WHILE and UNTIL clauses, the boolean-expression is evaluated each time the clause is encountered; that is, once per iteration. Evaluation at each iteration involves looking up the values of any references in the expression. Thus, the result can change with each evaluation.

The choice of WHILE or UNTIL is usually a matter of convenience—there is always a way to convert one into the other. For example, "WHILE bool-expr" is equivalent to "UNTIL NOT (bool-expr)".

NOTE

To terminate execution of a REPEAT (or COUNT) loop, enter CNTL-D at the console. The DEBUG-86 command currently executing halts wherever it happens to be; if you are executing, the current instruction is completed before the break. DEBUG-86 responds to the CNTL-D character with the asterisk prompt.

To return to RUN and then optionally to ISIS-II, enter CNTL-C.

Here are some brief examples of the REPEAT command.

Example 1. Generate an ASCII table similar to table 6-2:

```
DEFINE .TEMP = 40H
REPEAT
  WHILE .TEMP <= 7EH
  EVALUATE .TEMP
  .TEMP = .TEMP + 1
ENDR
```

Example 2. Single-step through the user program for each instruction until a repetitious routine (.DELAY) is reached:

```
REPEAT
  UNTIL CS:IP = .DELAY
  STEP
  ASM CS:IP
ENDR
```

Example 3. Using a complex combination of conditions in the boolean expression:

```
REPEAT
  UNTIL (CS:IP > .END XOR .VAR1 = 0) OR (.TEMP > 0 XOR .VAR2 = 1)
  STEP
  REGISTER
ENDR
```

Example 4. Execute from the start of the program (.START) until a breakpoint (.ERROR) is reached, display status registers, then continue execution, and displaying registers until a terminating condition (BYTE .VAR = 2) is reached:

```
REPEAT
  GO TILL .ERROR
  REGISTER
  UNTIL BYTE .VAR = 2
ENDR
```

COUNT Command

Like REPEAT, the COUNT command sets up a loop. In addition to the WHILE and UNTIL clauses discussed under REPEAT, COUNT includes a loop counter that terminates the loop if no exit condition is met before the counter runs out.

The COUNT command has the form:

```
COUNT arithmetic-expression<cr>
 [command<cr>
 [WHILE boolean-expression<cr>]   ...
 [UNTIL boolean-expression<cr>]
END<cr>
```

The *arithmetic-expression* after COUNT controls the (maximum) number of iterations to be performed. If a numeric constant is used (for example, COUNT 10), DEBUG-86 interprets it in implicit hexadecimal base; in other words, any number entered after COUNT without an explicit radix is interpreted as a hexidecimal number.

If the entry after COUNT is an *arithmetic-expression,* it is evaluated to give the number of iterations. The COUNT expression is evaluated *once,* before any loop elements are encountered. It is not evaluated again on any interation. The COUNT expression uses the values of any references it contains as they stand at the time of evaluation. For example, consider the following command sequence:

```
DEFINE .XX = 2
COUNT .XX
  .XX = .XX + 1
END
```

This loop goes through *two* iterations, although .XX has value 4 when the loop terminates.

The loop terminates when the number of iterations given by the COUNT expression has been performed *or* when an exit condition is tested and causes exit, *whichever comes first*. The following example illustrates this concept.

```
DEFINE .XX = 1
COUNT 5
  .XX = .XX + 1
  WHILE .XX < 5
END
```

To show that the loop terminates on the WHILE condition before the COUNT expression is exhausted, we can "track" the loop in operation. Table 6-9 shows the track.

Table 6-9. Tracking a COUNT Command

Iteration	.XX	.XX< 5
1	2	TRUE
2	3	TRUE
3	4	TRUE
4	5	FALSE

The loop terminates during the fourth iteration, when .XX < 5 becomes FALSE.

Conversely, the COUNT expression specifies the maximum number of iterations to be performed in case no exit clause produces an exit on any iteration. For example:

```
COUNT 10T
  UNTIL CS:IP = .DELAY
  STEP
  ASM CS:IP
END
```

In this command, the COUNT expression specifies a maximum of ten STEPs, in case the first instruction at .DELAY is not reached during any iteration.

With a REPEAT command or with a COUNT command that include one or more clauses, there may be no direct way to tell how many iterations occurred before the loop terminated. For these cases, you can insert a loop counter as a loop element. For example, to obtain table 6-9 as a display you could use the following sequence:

```
BASE = T
DEFINE .ITER = 0
DEFINE .XX = 1
COUNT 10T
  .XX = .XX + 1
  .ITER = .ITER + 1
  .ITER
  .XX
  BOOL .XX < 5
  WHILE .XX < 5
END
```

The command BOOL .XX < 5 produces a display of TRUE or FALSE.

The following example executes to a breakpoint, displays 8086 registers, then continues executing, breaking, and displaying for 10 iterations:

```
COUNT 10T
  GO TILL .PAUSE EXECUTED
  REGISTER
END
```

IF Command

The IF command permits conditional execution in a command sequence. The IF command has the form:

```
IF boolean-expression [THEN]<cr>
  [command<cr>] ...
[ORIF boolean-expression [THEN]<cr>] ...
  [command<cr>] ...
[ELSE<cr>
  [command<cr>]...]
END<cr>
```

The command must have the IF clause; the ORIF and ELSE clauses are optional. The command can include as many ORIF clauses as desired. The IF and ORIF clauses each contain a single condition (boolean expression). Any clause can contain none, one, or more commands. A clause with no commands simply produces an exit when its condition is TRUE.

DEBUG-86 examines each boolean expression in turn, clause by clause, looking for the first TRUE condition. If a TRUE condition is found, the commands in that clause are executed and the IF command terminates. If none of the conditions is TRUE, the commands in the ELSE clause are executed and the IF command terminates. If the ELSE clause is omitted and no condition is TRUE, the IF command terminates with no commands executed.

The END keyword is required to close off the IF command; it can be written as ENDIF to clarify nesting.

The following is an example of the IF command.

```
IP = 1
IF IP < 1
  EVALUATE 1
ORIF IP < 2
  EVALUATE 2
ORIF IP < 3
  EVALUATE 3
ELSE
  EVALUATE 4
END
```

This example displays the result of EVALUATE 2 and then terminates. The first condition (IF IP < 1) is FALSE, so EVALUATE 1 is skipped. The second condition (ORIF IP < 2) is TRUE, so EVALUATE 2 is executed and the IF command terminates. The third condition (ORIF IP < 3) is not tested, even though it happens to be TRUE.

In practice, however, the IF command is useful when it is nested in a REPEAT or COUNT loop rather than appearing at the top level. A nested IF command enables you to test conditions that can change (due to other commands in the loop), whereas when an IF command is at the top level the TRUE or FALSE state of any condition is known, or can be determined with the BOOL command. Thus, the result from the previous example can be obtained with fewer steps:

```
BOOL IP < 1  (Displays FALSE)
BOOL IP < 2  (Displays TRUE)
EVALUATE 2
```

Nesting Compound Commands

The REPEAT, COUNT, and IF commands can be nested to provide a variety of control structures.

Each nested compound command must have its own END keyword. When entering a nested command sequence, you may wish to use the keywords ENDR, ENDC, and ENDIF to help you keep straight which command you intend to close off. DEBUG-86 does not check nesting levels at entry, and if an END is omitted, the resulting error makes it necessary to enter the entire command again.

Each nested REPEAT or COUNT command can contain its own exit clauses (WHILE or UNTIL). Each exit clause can terminate the loop that contains it, but has no effect on any outer loops or commands.

As an example of nesting, suppose you want to STEP through a program with disassembled display, but skip a repetitive timeout routine, .DELAY, that is called

with an 8086 short-call instruction several times during program execution. One way to achieve this effect is with the following command sequence:

```
REPEAT
  IF CS:IP = .DELAY
   IP = WORD SS:SP
   SP = SP + 2
  ENDIF
  STEP
  ASM CS:IP
ENDR
```

At each call to .DELAY in the program, the displacement of the return address for the call is pushed on the stack. The keyword SP refers to the stack pointer, and SS is the stack segment register; SS:SP is the address of the top of the stack where the return address is stored. The effects of the commands IP = WORD SS:SP and SP = SP + 2 are to load the return address back into IP and reset the stack pointer just as if the return instruction at the end of .DELAY had been executed.

As another example of nesting, suppose the user code at statements #21 and #22 is incorrect or not written yet. The following sequence executes to the point where substitute code is to be inserted, inserts the code (equivalent to IF MARK > 0 THEN PTR = PTR + 2 in PL/M), then continues executing beginning with statement #23 (the insertion is made any time execution reaches statement #21):

```
GO FROM .START TILL #21
REPEAT
  IF WORD .MARK > 0
   WORD .PTR = WORD .PTR + 2
  ENDIF
  GO FROM #23
ENDR
```

An exit can be made only when a condition is *tested*, not when it occurs. To cause an exit, the test must be placed at the point in the loop where the condition occurs. For example, consider the following command sequence:

```
REPEAT
  UNTIL IP = 1000H
  STEP
ENDR
```

In this command the condition IP = 1000H is tested after every STEP. If the sequence of STEPs reaches IP = 1000H as the next instruction, the loop will terminate. By contrast, consider this example:

```
REPEAT
  UNTIL IP = 1000H
  COUNT 10
    STEP
  ENDC
ENDR
```

In the second example, the condition IP = 1000H is tested after every *ten* STEPs. The loop exits only if IP = 1000H occurs at the *end* of some group of ten instructions. If IP = 1000H occurs *during* one of the groups of ten STEPs, the loop does not terminate because that condition is changed by subsequent STEPs before the test can be made.

If the command has more than one exit clause, each exit clause is tested when it is encountered. If the result at the moment of the test causes an exit, the loop terminates; otherwise, the loop proceeds to the next element.

The loop exits only when the current test causes it, even though some other clause in the loop would cause an exit if it could be tested at that moment. Consider this (artificial) example:

```
DEFINE .ZZ = 0
CS = 780
IP = 0
REPEAT
  UNTIL IP > 10H
  COUNT 5
   STEP
  ENDC
  ASM CS:IP
  WHILE .ZZ = 0
  .ZZ = .ZZ + 1
ENDR
```

Assume for this example that the code being executed (with STEP) contains only two-byte instructions. Then, after the first time through the loop, IP = 0AH (10T) and .ZZ = 1. On the second iteration, the test IP > 10H is FALSE when it is encountered, so the STEP and ASM commands are executed again. At this point, IP > 10H is TRUE but since it is not tested, no exit occurs. Instead, the condition .ZZ = 0 is tested, found to be false, and the loop exits.

CHAPTER 7
ERROR MESSAGES

This chapter lists the error messages issued by ISIS-II, RUN, DEBUG-86, and nonresident system routines.

Error message numbers are allocated as follows:

- 1-99 inclusive — ISIS-II resident routines (8080/8085 mode)
- 100-119 inclusive — RUN command (8086 mode)
- 120-199 inclusive — DEBUG-86 (8086 mode)
- 200-255 inclusive — nonresident system routines (8080/8085 mode)

The following manual is referred to in this chapter as "reference 1":

Intellec Series III Microcomputer Development System Programmer's Reference Manual

ISIS-II Error Routines (8080/8085 Mode)

Errors encountered by ISIS-II are either fatal or nonfatal. In the following lists fatal errors are noted as such. The other errors are generally nonfatal unless they are issued by the CONSOL system call (see tables 7-1 and 7-2).

A nonfatal error immediately halts processing and permits your program to take a recovery path of your choosing. The error number is returned to your program.

If an error occurs when you are entering a console command, the error is echoed followed by an error message. For example, the following input results in the error message shown:

```
-COPY #3:PR:CREDIT TO #3:<CR>
:PR:CREDIT, UNRECOGNIZED DEVICE NAME
```

A fatal error immediately halts processing but does not permit recovery. Control returns to ISIS-II which overlays some user program area with nonresident ISIS-II files, and displays the following error message:

```
ERROR nnn USER PC mmmm
```

where nnn is the error number and mmmm is the contents of the program counter when the error occurred.

In general, after displaying an error message, the system displays the ISIS-II prompt character (a hyphen) and waits for you to enter the corrected input.

The action taken in response to fatal errors depends on the setting of an internal system switch called the debug toggle. That switch indicates whether control is to return to ISIS-II (debug=0) or the Monitor (debug=1) when an error occurs.

Any of the following actions sets the debug toggle to one and transfers control to the Monitor:

- Pressing interrupt switch 0 while a program is running.
- Executing program load with the DEBUG switch specified in the command line.
- Executing a LOAD system call with a transfer value of 2.

Any of the following actions sets the debug toggle to zero, performs the operation listed, then transfers control to ISIS-II:

- Pressing interrupt switch 1 while a program is running. This action terminates processing.
- Executing an EXIT system call. This action terminates a program.
- Executing a LOAD system call with a transfer value of 1. This action loads an absolute object file.
- Executing a Monitor G8 command. This action exits the Monitor.

If the debug toggle is zero when a fatal error occurs, the following occur:

- All open files are closed in their current state, including :CI: and :CO:.
- The initial system console device is opened as :CI: and :CO:.
- A fresh copy of ISIS-II is read in from the disk, and ISIS-II prompts for a command with a hyphen (-).

If the debug toggle is set to one when a fatal error occurs, the following occur:

- All open files are left open.
- Control passes to the Monitor.
- Monitor prompts for a command with a period (.).

At this point Monitor commands can be used to examine registers and memory to try to determine the cause of the error. However, the program should not be restarted with a simple Monitor G command, because the ISIS-II restart address has not been saved. DO NOT RESET THE SYSTEM AT THIS POINT. A G8 command should be used instead so all files are closed. Rebooting does not close files.

NOTE

Although programs cannot be loaded in the ISIS-II area, the ISIS-II area is not protected from a running program. If a program should happen to destroy parts of ISIS-II, subsequent system calls may not operate correctly and input/output may destroy areas on your disk. This would happen mainly when an undebugged program is running. ISIS-II can always be restored by bootstrapping from a good system disk.

Table 7-1. Nonfatal Error Numbers Returned by System Calls

OPEN	3, 4, 5, 9, 12, 13, 14, 22, 23, 25, 28.
READ	2, 8.
WRITE	2, 6.
SEEK	2, 19, 20, 27, 31, 35.
RESCAN	2, 21.
CLOSE	2.
DELETE	4, 5, 13, 14, 17, 23, 28, 32.
RENAME	4, 5, 10, 11, 13, 17, 23, 28.
ATTRIB	4, 5, 13, 23, 26, 28.
CONSOL	None; all errors are fatal.
WHOCON	None.
ERROR	None.
LOAD	3, 4, 5, 12, 13, 22, 23, 28, 34.
EXIT	None.
SPATH	4, 5, 23, 28.

Table 7-2. Fatal Errors Issued by System Calls

OPEN	1, 7, 24, 30, 33.
READ	24, 30, 33.
WRITE	7, 24, 30, 33.
SEEK	7, 24, 30, 33.
RESCAN	33.
CLOSE	33.
DELETE	1, 24, 30, 33.
RENAME	1, 24, 30, 33.
ATTRIB	1, 24, 30, 33.
CONSOL	1, 4, 5, 12, 13, 14, 22, 23, 24, 28, 30, 33.
WHOCON	33.
ERROR	33.
LOAD	1, 15, 16, 24, 30, 33.
SPATH	33.

ISIS-II error messages codes are:

1. Fatal error. The memory area from 3000H to program origin is used for input/output buffers. Too few buffers were allocated to meet the current request in addition to earlier requests. See reference 1 for information on how to allocate buffers.
2. Illegal AFTN argument. The number supplied as an AFTN (active file table number) is inappropriate. Perhaps your program closed a file prematurely and then tried to read it. Active file table numbers are described in reference 1.
3. Fatal error. AFT (Active File Table) is full. At most, six files may be active at one time. You must close one of your open files before a file can successfully be opened. (Reference 1)
4. Incorrectly specified filename. You have possibly entered too many characters for filename, as in OLDFILE.1 (the maximum is six characters before the period, three after). Filename conventions are described in Chapter 3.
5. Unrecognized device name. You have entered an incorrect device name, as in :PR: for the line printer :LP:. Check the device names in Chapter 3.
6. Attempt to write to input device. An attempt has been made to write to an input device. You can only write to an output device, such as a line printer (:LP:). See Chapter 3 for information on devices.
7. Fatal error. The disk is full. Check that you have specified the intended disk.
8. Attempt to read from output device. Some devices, like the line printer (:LP:), are output only and cannot be read. The current operation either should not be a READ or needs to use a different device name. See Chapter 3 for devices.
9. Disk directory is full. There is no room on the target disk's directory to add an additional filename. The limit is 200 entries for flexible disks and 992 entries for hard platters.
10. Pathname is not on same disk. A system call was attempted (RENAME) that requires two pathnames on the same device but the specified pathnames did not specify the same device. (Reference 1)
11. File already exists. A filename identical to the one just used was found. Perhaps a different drive was intended, or a different spelling of the filename.
12. File is already open. Only console input (:CI:) and console output (:CO:) may be opened multiple times. If the spelling of the filename is correct, a flaw may exist in the program logic. For example, an earlier module may be using the file too soon or there may be an unintended loop.

13. No such file. The specified filename could not be found in the directory on the disk in the drive indicated by your command. A different drive or disk may have the file. For example, a console request to load a RUN file with a default extension of .86.
14. Write-protected file encountered. The intended operation (e.g., WRITE, RENAME, DELETE) could not be done because the specified file has the write-protect or format attribute set.
15. Fatal error. ISIS overwrite. The system detected an attempt to write into the area reserved for the ISIS resident files, i.e., below 3000H. Such an operation would create unpredictable results and is disallowed.
16. Fatal error. Bad load format. This error was possibly caused by a source-language file. Files to be loaded for 8080/8085 execution must be in absolute object module format.
17. Not a disk file. An attempt was made to reference a disk file on a wrong device type, with an improper pathname, such as :HP:FILE2 instead of :Fn:FILE2. File accessing conventions are described in Chapter 3.
18. Illegal ISIS commands. This error results when an ISIS system call is made with an illegal command number.
19. Attempted seek on non-disk file. Seeks on physical devices other than disk drives are invalid (:BB: is an exception and is valid). (Reference 1 gives information on the SEEK system call.)
20. Attempted back seek too far. The seek attempted to go beyond the beginning of the file; MARKER is set to zero. (Reference 1)
21. Can't rescan. The file was not opened for line-editing. (Reference 1)
22. Illegal access mode to open. Only 1, 2, and 3 are valid, meaning input (read), output (write), or update (both read and write). (Reference 1)
23. Missing filename. The system expected a filename, but one was not supplied.
24. Fatal error. Disk input/output hardware error. When error number 24 occurs, an additional message is displayed:

```
STATUS=00nn
D=x T=yyy S=zzz
```

where x represents the drive number, yyy the track address, zzz the sector address, and where nn has the following meanings:

For flexible disks:

01	Deleted record
02	Data field CRC error
03	Invalid address mark
04	Seek error
08	Address error
0A	ID field CRC error
0E	No address mark
0F	Incorrect data address mark
10	Data overrun or data underrun
20	Attempt to write on Write Protect
40	Drive has indicated a Write error
80	Drive not ready

For hard disks:

01	ID field miscompare
02	Data field CRC error
04	Seek error
08	Bad sector address
0A	ID field CRC error
0B	Protocol violations
0C	Bad track address
0E	No ID address mark or sector not found
0F	Bad data field address mark
10	Format error
20	Attempt to write on write-protected drive
40	Drive has indicated a write error
80	Drive not ready

25. Illegal echo file. An echo file must have an active file table number (AFTN) between 0 an 255, and must already be opened for output. Check that these conditions are met. (Reference 1)
26. Illegal attribute identifier. This error refers to the second parameter to the ATTRIB system call routine. Check that you have specified a valid parameter. Only 0, 1, 2, or 3 is valid, meaning the invisible, system, write-protect, or format attributes, respectively. (Reference 1)
27. Illegal seek command. An unsupported mode for the specified device was used in a seek command. (Reference 1)
28. Missing extension. An expected file extension was not supplied.
29. Fatal error. Premature EOF. An unexpected end of file was encountered from the console.
30. Fatal error. Drive specified was not ready.
31. Can't seek on write only file. Seeks can be executed only on read or update files. (Reference 1)
32. Can't delete open file. You need to close the file before attempting to delete it. Verify the pathname. (Reference 1)
33. Fatal error. Illegal system call parameter. A parameter was specified in a system call which is meant to be used as a pointer to a memory area intended for the receipt of data; however, ISIS found that this pointer was pointing to the memory space which ISIS occupies. ISIS will not allow a user to write into its memory space. (Reference 1)
34. Fatal error. The return switch in a LOAD system call was not 0, 1 or 2, the only valid values. (Reference 1)
35. Seek past EOF. An attempt was made to extend a file opened for input by seeking past end-of-file. (Reference 1)

RUN Program Error Messages (8086 Execution Mode)

When an error occurs under the RUN program, both the error number and the error message are displayed. Processing halts and control returns to the RUN program. The system displays the RUN prompt (>) and waits for you to enter a new command.

Errors 117 through 119 are warnings. The system displays the warning message and processing continues.

101. HARDWARE NOT RESPONDING (fatal error)
     8086 hardware is not present or is malfunctioning.

102. INVALID SYNTAX
     RUN does not understand your request. Check the input line and re-enter.

103. COMMAND LINE TOO LONG
     The RUN activation line exceeds 120 characters.

104. INSUFFICIENT MEMORY TO LOAD
     The loader does not have enough memory to load the requested object file.

105. MISMATCHED SOFTWARE/FIRMWARE
     The version of RUN does not correctly operate with the 8086 firmware.

107. ILLEGAL LOAD ADDRESS
     Loader tried to load a user program into 8086 system memory. User 8086 memory begins at 7800H.

108. INVALID OBJECT FILE
     The file specified in a LOAD command is not a valid object file.

117. UNRESOLVED SYMBOLS (warning)
     The file just loaded contains externals that were not satisfied at link time. The file was loaded correctly except for references to the unsatisfied externals.

118. RAM FAILURE (warning)
     RAM failure was detected on the 86 processor board.

119. ROM CHECKSUM ERROR (warning)
     ROM checksum error was detected on the 86 processor board.

DEBUG-86 Error Messages (8086 Execution Mode)

When an error occurs under DEBUG-86, the system displays the error number only, as follows:

```
ERROR xxx
```

where xxx is a decimal error number.

After the error number, the system displays the DEBUG-86 prompt (*) and waits for you to enter a new command.

The following list defines DEBUG-86 error numbers:

120. Syntax error. The command line entered does not conform to the defined syntax.
121. Invalid token. The command line contains a token that does not follow the rules for a well-formed token.
122. No such line. The specified line number does not exist in the current module.
123. Inappropriate number. The value entered is not appropriate in the current context.
124. Partition bounds error. The partition values entered in a command are not correct. Either the left part of the partition is greater than the right part or the values of the partition extremes are out of range in the current context.
125. Symbol already exists. The symbol entered in a DEFINE command is already defined in the symbol table.
126. Symbol does not exist. The symbol referenced does not reside in the symbol table.
127. Memory failure. Data written to memory was not correctly read back. This error can be caused by writing data into non-existent or bad memory.
133. Null string error. A null string was used where a non-null string is required.
134. Memory overflow. Memory requirements of all dynamic tables exceed the amount of memory available. This error can be caused by an object module with too many symbols or line numbers.
135. Stack overflow. The capacity of a statically allocated stack internal to the diagnostic program has been exceeded.
136. Command too complex. The command complexity requires more memory than can be allocated.
137. Module does not exist. The specified module does not exist in the symbol table.
139. Excessive data. The amount of data to be inserted into a partition exceeds the size of the partition.
141. Unsuitable execute file. The file referenced in an execute command either contains code that is out-of-bounds for the execute command, or it is a main module.
142. Line too long. The command line was longer than 120 characters on a line.
143. Too many partitions. Number of partitions entered exceed the acceptable maximum of 19.
147. Pointer value required. A non-pointer value was used in a context that requires a pointer.
148. Integer value required. A non-zero base value was used in a context that requires an integer.
149. Differing bases. Two pointers with different bases were used in a context where pointers with the same base are required, e.g., the lower and upper bounds of a partition.

Console Command Interface Errors (8080/8085 Execution Mode)

201. Unrecognized switch. Certain predefined switches (e.g., P, S, A, U, C) can be used depending on the ISIS-II command. Some commands that have switches are IDISK, FORMAT, and COPY. Check the specified command in Chapter 4.
202. Unrecognized delimiter A character was encountered that was invalid in a name and not known as a delimiter.
203. Invalid syntax. There is an error in the command as entered. The error may be an unrecognized keyword or a missing comma, for example.
206. Illegal disk label. The label supplied violates the rules for a valid disk label.
208. Checksum error. The bits of the records read do not add up properly. An inappropriate input or medium was supplied. There may be an error in the internal format of the specified file that may have occurred during translation or linking. Retranslate and relink the source module.
209. Relo file sequence error. An inappropriate input file was specified.
210. Insufficient memory. The required amount of RAM is not present.
211. Record too long. A record longer than allowed was encountered.
212. Illegal relo type. Relocation types must conform to Intel standard formats.
213. Fixup bounds error. The required address violated numeric bounds on addresses.
214. Illegal SUBMIT parameter. An error was made in the actual parameter to be substituted for a formal parameter in a command sequence file. (See the SUBMIT command in Chapter 4.)
215. Argument too long. The number of characters in the actual argument must not exceed 31.
216. Too many parameters. More parameters were supplied than defined.
217. Object record too short. This error may be caused by an I/O error in the file to be loaded.
218. Illegal record format. The record format did not match the Intel standard.
219. Phase error. The expected phase input (e.g., for the next step of a translation process) was not correctly supplied.
220. No end-of-file record in object module file. There is an error in the internal format of the specified file. Retranslate and relink the source module.
221. Segment overflow during LINK operation. The output segment cannot be greater than 64k bytes.
222. Unrecognized record in object module file. There is an error in the internal format of the specified file. Retranslate and relink the source module.
223. Fixup record pointer is incorrect. There is an error in the internal format of the specified file. Retranslate and relink the source module.
224. Illegal record sequence in object module file in LINK. There is an error in the internal format of the specified file that may have occurred during translation. Retranslate and relink the source module.
225. Illegal module name specified. An illegal or misspelled module name was entered.
226. Module name exeeds 31 characters. Module names exeeding 31 characters may not be used.
227. Command syntax requires left parenthesis. There is a missing left parenthesis in the command line. Re-enter the command correctly.

228. Command syntax requires right parenthesis. There is a missing right parenthesis in the command line. Re-enter the command correctly.
229. Unrecognized control specified in command. A character string other than the expected control keyword was entered. Enter the correct control keyword.
230. Duplicate symbol found. You have attempted to add a symbol that already exists.
231. File already exists. The file specified in a CREATE command already exists.
232. Unrecognized command. An illegal or misspelled command was entered.
233. Command syntax requires a TO clause. The command syntax requires a TO clause to specify the output file.
234. Filename illegally duplicated in command. The same filename is specified both as an input and output file.
235. File specified in command is not a library file. The specified file is not a library file.
236. More than 249 common segments in input files. You cannot have more than 249 common segments.
237. Specified common segment not found in object file. The input module does not contain the common segment specified in the command.
238. Illegal stack content record in object file. There is an error in the internal format of the specified file that may have occurred during the translation and link process. Retranslate and relink the source module.
239. No module header in input object file. There is an error in the internal format of the specified file. Retranslate and relink the source module.
240. Program exeeds 64k bytes. The output module to be placed in the output file exeeds the maximum of 64k bytes.

Other Console Command Interface Errors

Additional 8080/8085 link and locate error messages are described in the *MCS-80/85 Utilities User's Guide for 8080/8085-Based Development Systems*.

All 8086 mode link and locate error messages that may occur during LINK86, LOC86, OH86, and LIB86 operations are described in the *iAPX Family Utilities User's Guide for 8086-Based Development Systems*.

APPENDIX A
HEXADECIMAL PAPER TAPE FORMAT

Object code is stored on paper tape in an ASCII representation of the program in memory. The code is blocked into records, each of which contains the record type, length, type, memory load address, and checksum in addition to the data. Figure A-1 shows the frames of a tape record.

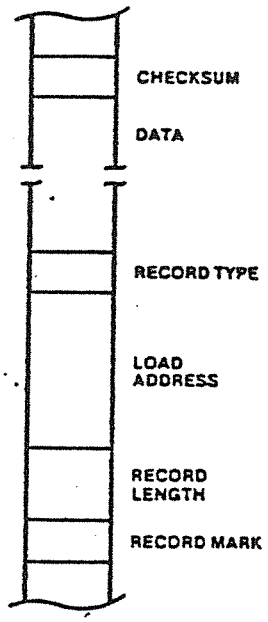


Figure A-1. Paper Tape Record Format

The Record Mark is a colon (3AH) and is used to signal the start of a record.

The Record Length is the count of the data bytes in the record. A record length of zero indicates end-of-file.

The Load Address specifies the address at which the first data byte will be loaded. The successive data bytes will be stored in successive memory locations.

The Record Type specifies the type of this record. All data records are type 0. End-of-file records can be type 0 or 1.

The Data consists of two frames per memory word. The data is represented by hexadecimal values 00H through FFH.

The Checksum is the negative of the sum of all 8-bit bytes in the record, beginning with the Record Length and ending with the last Data byte, evaluated modulo 256. The sum of all bytes in the record (including the checksum) should be zero.

APPENDIX B
HEXADECIMAL-DECIMAL CONVERSION

The following table is for hexadecimal to decimal and decimal to hexadecimal conversion. To find the decimal equivalent of a hexadecimal number, locate the hexadecimal number in the correct position and note the decimal equivalent. Add the decimal numbers.

To find the hexadecimal equivalent of a decimal number, locate the next lower decimal number in the table and note the hexadecimal number and its position. Subtract the decimal number from the table from the starting number. Find the difference in the table. Continue this process until there is no difference.

BYTE				BYTE			
HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC
0	0	0	0	0	0	0	0
1	4,096	1	256	1	16	1	1
2	8,192	2	512	2	32	2	2
3	12,288	3	768	3	48	3	3
4	16,384	4	1,024	4	64	4	4
5	20,480	5	1,280	5	80	5	5
6	24,576	6	1,536	6	96	6	6
7	28,672	7	1,792	7	112	7	7
8	32,768	8	2,048	8	128	8	8
9	36,864	9	2,304	9	144	9	9
A	40,960	A	2,560	A	160	A	10
B	45,056	B	2,816	B	176	B	11
C	49,152	C	3,072	C	192	C	12
D	53,248	D	3,328	D	208	D	13
E	57,344	E	3,548	E	224	E	14
F	61,440	F	3,840	F	240	F	15

APPENDIX C
ASCII CODES

Table C-1. ASCII Code List

Decimal	Octal	Hexadecimal	Character
0	000	00	NUL
1	001	01	SOH
2	002	02	STX
3	003	03	ETX
4	004	04	EOT
5	005	05	ENQ
6	006	06	ACK
7	007	07	BEL
8	010	08	BS
9	011	09	HT
10	012	0A	LF
11	013	0B	VT
12	014	0C	FF
13	015	0D	CR
14	016	0E	SO
15	017	0F	SI
16	020	10	DLE
17	021	11	DC1
18	022	12	DC2
19	023	13	DC3
20	024	14	DC4
21	025	15	NAK
22	026	16	SYN
23	027	17	ETB
24	030	18	CAN
25	031	19	EM
26	032	1A	SUB
27	033	1B	ESC
28	034	1C	FS
29	035	1D	GS
30	036	1E	RS
31	037	1F	US
32	040	20	SP
33	041	21	!
34	042	22	"
35	043	23	#
36	044	24	$
37	045	25	%
38	046	26	&
39	047	27	'
40	050	28	(
41	051	29	)
42	052	2A	*
43	053	2B	+
44	054	2C	,
45	055	2D	-
46	056	2E	.
47	057	2F	/
48	060	30	0
49	061	31	1
50	062	32	2
51	063	33	3
52	064	34	4
53	065	35	5
54	066	36	6
55	067	37	7
56	070	38	8
57	071	39	9
58	072	3A	:
59	073	3B	;
60	074	3C	<
61	075	3D	=
62	076	3E	>
63	077	3F	?

Decimal	Octal	Hexadecimal	Character
64	100	40	@
65	101	41	A
66	102	42	B
67	103	43	C
68	104	44	D
69	105	45	E
70	106	46	F
71	107	47	G
72	110	48	H
73	111	49	I
74	112	4A	J
75	113	4B	K
76	114	4C	L
77	115	4D	M
78	116	4E	N
79	117	4F	O
80	120	50	P
81	121	51	Q
82	122	52	R
83	123	53	S
84	124	54	T
85	125	55	U
86	126	56	V
87	127	57	W
88	130	58	X
89	131	59	Y
90	132	5A	Z
91	133	5B	[
92	134	5C	\
93	135	5D	]
94	136	5E	^
95	137	5F	_
96	140	60	`
97	141	61	a
98	142	62	b
99	143	63	c
100	144	64	d
101	145	65	e
102	146	66	f
103	147	67	g
104	150	68	h
105	151	69	i
106	152	6A	j
107	153	6B	k
108	154	6C	l
109	155	6D	m
110	156	6E	n
111	157	6F	o
112	160	70	p
113	161	71	q
114	162	72	r
115	163	73	s
116	164	74	t
117	165	75	u
118	166	76	v
119	167	77	w
120	170	78	x
121	171	79	y
122	172	7A	z
123	173	7B	{
124	174	7C	|
125	175	7D	}
126	176	7E	~
127	177	7F	DEL

Table C-2. ASCII Code Definition

Abbreviation	Meaning	Decimal Code
NUL	NULL Character	0
SOH	Start of Heading	1
STX	Start of Text	2
ETX	End of Text	3
EOT	End of Transmission	4
ENQ	Enquiry	5
ACK	Acknowledge	6
BEL	Bell	7
BS	Backspace	8
HT	Horizontal Tabulation	9
LF	Line Feed	10
VT	Vertical Tabulation	11
FF	Form Feed	12
CR	Carriage Return	13
SO	Shift Out	14
SI	Shift In	15
DLE	Data Link Escape	16
DC1	Device Control 1	17
DC2	Device Control 2	18
DC3	Device Control 3	19
DC4	Device Control 4	20
NAK	Negative Acknowledge	21
SYN	Synchronous Idle	22
ETB	End of Transmission Block	23
CAN	Cancel	24
EM	End of Medium	25
SUB	Substitute	26
ESC	Escape	27
FS	File Separator	28
GS	Group Separator	29
RS	Record Separator	30
US	Unit Separator	31
SP	Space	32
DEL	Delete	127

APPENDIX D
SUMMARY OF ISIS-II CONSOLE COMMANDS

This appendix provides a summary of command syntax for ISIS-II commands.

Disk Maintenance Commands

IDISK—Disk Formatting Command

```
IDISK :Fn:label [switches]<cr>
```

where

label is the name to be given to the blank disk.

switches are one or more of the following:

S	Formats the new disk as a basic system disk.
P	Specifies single drive mode.
FROM n	Specifies the disk drive containing the source disk files needed for formatting the new disk. The value n is an integer 0-9 for drive numbers 0 through 9. If the FROM n switch is not specified, the default is to drive 0.

FORMAT—Disk Formatting Command

```
FORMAT :Fn:label [switches]<cr>
```

where

label is the name to be given to the disk.

switches are one or more of the following:

A	Copies all files to the specified disk except files (other than ISIS-II system format files) with the format attribute set.
S	Copies the basic format files and all files with the system attribute set.
FROM n	Specifies the disk drive containing the disk files needed for formatting. n is an integer 0-9, for drives 0 through 9. If the FROM n switch is not specified, the default is to drive 0.

FIXMAP—Hard Disk Mapping Command

```
FIXMAP drive<cr>
```

where

drive is the number of the hard disk drive on which the command is to operate; *drive* is an integer value of 0 or 1.

FIXMAP Commands

Mark *disk-address*	Change the known state of a sector from good to bad.
Free *disk-address*	Change the known state of a sector from bad to good.
List *filename*	List all known bad sectors.
Count	List the number of known bad sectors.
Record	Record changes specified by Mark and Free.
Quit	Exit to ISIS-II without recording changes.
Exit	Record changes and exit to ISIS-II.

A *disk-address* takes the form:

```
track,sector[T]
```

where

track is a number from 0 to 199 that specifies the logical track address containing the bad sector.

sector is a number from 1 to 144 that specifies the logical sector address of the bad sector within the track.

T is an optional switch indicating that a group of 36 sectors should be processed.

File Control Commands

DIR—Disk Directory Listing

```
DIR [FOR filename][TO listfile][switches]<cr>
```

where

filename is the file (or group of files specified with the wild card construction) whose directory entry is to be listed.

switches are one or more of the following:

0-9	Lists the directory of the disk in :F0:, :F1:, :F2:, ... :F9:.
I	Lists all files, including files with the invisible attribute set.
F	Gives fast output, listing only filenames.
O	Prints the directory in a single column format.
Z	Prints the number of sectors presently used.
P	Specifies single drive mode.

COPY—Copy a File

```
COPY [:Fn:]infile [,...] TO {[:Fn:][outfile]} [switches]<cr>
                            {:device:        }
```

where

infile is a file (or group of files when using the wild card construct) to be copied.

outfile is a file to be created or recreated.

:device: is an output device, such as :LP:, :TO:, :HP:, or :CO:

switches are one or more of the following:

S	Copies files with the system attribute set.
N	Copies files without the system or format attribute set.
P	Specifies single drive mode.
Q	Specifies the query mode.
C	Creates *outfile* with the attributes set from the infile.
B	Deletes an existing file without displaying the "ALREADY EXISTS" prompt.
U	Opens *outfile* for update instead of deleting it.

HDCOPY—Copy Hard Disk Tracks

```
HDCOPY {indrive TO outdrive} <cr>
       {      BACKUP       }
```

where

indrive is the number of the drive containing the source hard disk.

outdrive is the number of the drive containing the destination hard disk.

Both drive numbers must be 0 or 1, but both cannot be the same drive number. Both drives must be a hard disk drive.

BACKUP is a switch that can be used to backup a removable hard disk platter.

DELETE—Delete a Disk File

```
DELETE [:Fn:]filename [Q] [, . . . [Q]] [P]<cr>
```

where

filename is the name of a file to be deleted. The wild card construction can be used to delete a group of files.

Q	Specifies the query mode.
P	Specifies single drive mode.

RENAME—Rename a Disk File

```
RENAME [:Fn:]oldname TO [:Fn:]newname<cr>
```

where

:Fn: must be the same for both *oldname* and *newname*.

oldname is the name of an existing file whose write-protect or format attribute is not set.

newname is the new name to be assigned to *oldname*.

ATTRIB—Change/Display Disk File Attributes

```
ATTRIB [:Fn:]filename [attriblist] [Q]<cr>
```

where

filename is a disk file whose attributes are to be changed. The wild card construction can be used to change and/or display the attributes of a group of files.

attriblist is one or more of the following:

I0 or I1	Resets (I0) or sets (I1) the invisible attribute.
W0 or W1	Resets (W0) or sets (W1) the write-protect attribute.
F0 or F1	Resets (F0) or sets (F1) the format attribute.
S0 or S1	Resets (S0) or sets (S1) the system attribute.
Q	Specifies query mode operation.

8080/8085 Program Execution Commands

Filename—Direct Program Execution

```
[:Fn:]filename [parameters]
```

where

parameters are parameters needed by filename.

DEBUG—Transfer Control to Monitor

```
-DEBUG [[:Fn:]filename [parameters]]<cr>
```

where

filename is any ISIS-II command or the file name of an executable program.

parameters are the normal parameters of the program to be executed.

SUBMIT—Non-Interactive Program Execution

```
SUBMIT [:Fn:]filename[(parameter[, . . .])9]<cr>
```

where

filename is the name (and extension, if any) of the file containing the command sequence definition (explained below). If extension is omitted, SUBMIT assumes the default extension *.CSD*.

parameter is an actual value that is to replace a formal parameter in the command sequence definition file. The maximum number of parameters allowed is 10. If you omit a parameter from the SUBMIT list, enter a comma in its place.

A parameter is a character string of up to 31 characters.

ATTRIB—Change/Display Disk File Attributes

```
ATTRIB [:Fn:]filename [attriblist] [Q]<cr>
```

where

filename is a disk file whose attributes are to be changed. The wild card construction can be used to change and/or display the attributes of a group of files.

attriblist is one or more of the following:

I0 or I1	Resets (I0) or sets (I1) the invisible attribute.
W0 or W1	Resets (W0) or sets (W1) the write-protect attribute.
F0 or F1	Resets (F0) or sets (F1) the format attribute.
S0 or S1	Resets (S0) or sets (S1) the system attribute.
Q	Specifies query mode operation.

8080/8085 Program Execution Commands

Filename—Direct Program Execution

```
.[:Fn:]filename [parameters]
```

where

parameters are parameters needed by *filename*.

DEBUG—Transfer Control to Monitor

```
-DEBUG [[:Fn:]filename [parameters]]<cr>
```

where

filename is any ISIS-II command or the file name of an executable program.

parameters are the normal parameters of the program to be executed.

SUBMIT—Non-Interactive Program Execution

```
SUBMIT [:Fn:]filename[(parameter[, . . .])9]<cr>
```

where

filename is the name (and extension, if any) of the file containing the command sequence definition (explained below). If extension is omitted, SUBMIT assumes the default extension *.CSD*.

parameter is an actual value that is to replace a formal parameter in the command sequence definition file. The maximum number of parameters allowed is 10. If you omit a parameter from the SUBMIT list, enter a comma in its place.

A parameter is a character string of up to 31 characters.

8086 Program Execution Commands

RUN—Activate 8086 Execution Mode

```
RUN [[:Fn:]filename [parameters][;comments]]<cr >
```

where

filename is the name of your 8086 program. If you enter no extension, the system assumes a default extension of .86.

comment is one or more ASCII characters not including a carriage return or line feed. Comments always begin with a semicolon.

WORK—Change Default Drive for Workfiles

```
[RUN] WORK [:Fn:]<cr>
```

where

:Fn: specifies the drive n that is to be set as the default drive for your temporary workfiles. n is an integer value between 0 and 9 inclusive. The initial system default is :F1: If :Fn: is not specified, the current default is displayed.

DATE—Change/Display System Date

```
[RUN] DATE [nn/nn/nn]<cr>
```

where

nn is any integer value between 00 and 99 inclusive that specifies the date desired. If the date is not specified, the last date entered is displayed. The initial default date is 09/01/80.

EXIT

```
EXIT<cr>
```

APPENDIX E
SUMMARY OF MONITOR COMMANDS

This appendix provides a summary of the command syntax for Monitor commands.

Program Execution Commands

G—Execute Command

```
G[start-address][breakpoint1][,breakpoint2]<cr>
```

Monitor I/O Configuration Commands

A—Assign Command

```
Alogical-device=physical-device <cr>
```

The possible values of *physical-device* for each *logical-device* are:

Logical Device	Physical Device
CONSOLE	T or TTY (teletype terminal) C or CRT (compatible CRT terminal) B or BATCH (batch mode) 1 (user-defined device for which a user-written program is present)
READER	T or TTY (teletype terminal) P or PTR (high speed paper tape reader) 1 or 2 (user-defined devices for which user-written driver programs are present)
PUNCH	T or TTY (teletype terminal) P or PTP (high speed paper tape punch) 1 or 2 (user-defined devices on which user-written driver programs are present)
LIST	T or TTY (teletype terminal) C or CRT (compatible CRT terminal) L or LPT (line printer) 1 (user-defined device for which a user-written driver program is present)

Q—Query Command

```
Q<cr>
```

Memory Control Commands

D—Display Command

```
Dstart-address,end-address<cr>
```

F—Fill Command

```
Fstart-address,end-address,constant<cr>
```

M—Move Command

```
Mstart-address,end-address,destination-address<cr>
```

S—Substitute Command

```
Saddress,[databyte][,[databyte]][...]<cr>
```

X—Register Command (Display Form)

```
X<cr>
```

X—Register Command (Modify Form)

```
Xregister, [data] [, data]] [,...]<cr>
```

Paper Tape I/O Commands

R—Read Command

```
Rbias<cr>
```

W—Write Command

```
Wstart-address,end-address<cr>
```

E—End-of-File Command

```
Eentry-point<cr>
```

N—Null Command

```
N<cr>
```

Utility Command

H—Hexadecimal Command

Hnumber1,number2<cr>

APPENDIX F
SUMMARY OF DEBUG-86 COMMANDS

This appendix provides a summary of the command syntax of DEBUG-86 commands.

Utility Commands

DEBUG-86 utility commands provide file management capabilities. The utility commands are:

DEBUG—Transfer Control to DEBUG-86

```
[:Fn:]RUN DEBUG [[:Fn:]filename [parameters]]
```

where

filename is the name (including extension) of a program that is a valid absolute, PIC, or LTL 8086 object module. If no extension is specified, RUN adds an extension of .86. If filename ends with a period (as in MYPROG.), the null extension is assumed.

parameters is a series of one or more ASCII characters (separated by commas or spaces) representing variable data required by the user program and to be processed by the program.

EXIT—Exit DEBUG-86

```
EXIT<cr>
```

LOAD—Load 8086 Object Code

```
LOAD [:Fn:]filename [{NOSYMBOL | NOLINE}...] <cr>
```

where

filename is the complete name of a disk file that is a valid absolute, PIC or LTL 8086 object module. A default extension is not assumed.

NOSYMBOL is a modifier that prevents the program symbol table from being loaded.

NOLINE is a modifier that prevents the program line number table (in PL/M-86 or PASCAL-86 programs) from being loaded.

Execution Commands

GO—Execute 8086 Instructions

```
GO [FROM address] { [FOREVER]                                          } <cr>
                  { [TILL break-address [OR break-address]]            }
                  { [TILL break-register [OR break-register]]          }
```

where

FROM *address* specifies the address of the first instruction to be executed. If FROM *address* is omitted, execution begins at the address in the IP and CS. The *address* must specify the CS:IP content in the form nnnn:nnnn, as in 800:0 (leading zeros need not be entered).

break-address is an integer expression entered as a pointer that references a 20-bit execution address.

break-register is one of the breakpoint registers, BR0 or BR1. The address for BR forms a 20-bit memory address where DEBUG-86 writes a one-byte interrupt to get control.

GR Command

Display form:

```
GR<cr>
```

Change form:

```
GR = { FOREVER                                          } <cr>
     { TILL break-address [OR break-address]            }
     { TILL break-register [OR break-register]          }
```

where

break-address is an integer expression entered as a pointer that references a 20-bit execution address. The first *break-address* sets the contents of BR0; the second *break-address* sets the contents of BR1.

break-register is one of the breakpoint registers, BR0 or BR1 (or BR to denote both breakpoint registers), that is to be enabled.

STEP—Execute a Single Instruction

```
STEP [FROM address]<cr>
```

where

FROM *address* specifies the address where single step execution is to begin. If FROM *address* is omitted, the address in the IP and CS is used. The *address* must specify the CS:IP contents in the form nnnn:nnnn, as in 800:0 (leading zeros need not be entered).

Change Commands

Change Register—Change Content of a Register

register = *change-exp*<cr>

where

register is one of the following keyword references:

8086 Register Type	Keyword References
8-bit registers	RAL, RAH, RBL, RBH, RCL, RCH, RDL, RDH
16-bit registers	RAX, RBX, RCX, RDX, SP, BP, SI, DI, SS, CS, DS, ES, IP, RF
1-bit status flags	CFL, PFL, AFL, ZFL, SFL, TFL, IFL, DFL, OFL

change-exp is a numeric expression specifying the new contents of *register*.

Change Memory—Change Contents of Memory Locations

memory-type address {[TO *end-address*] / [LENGTH n]} = *change-exp* [,...]19<cr>

where

memory-type is one of the following keywords: BYTE, WORD, SINTEGER, INTEGER, POINTER.

address is a memory location entered as a pointer value containing a base (which can be omitted if 0) and a displacement. If a range is accessed, *address* is the starting address.

TO *end-address* specifies the upper limit of a range of memory that is to be modified. The *end-address* must be greater than or equal to *address*. Both addresses must have the same base.

LENGTH n specifies the number of bytes, words, or pointers (depending on *memory-type*) to be modified. The value n must be an integer value.

change-exp is the value to replace the contents of the specified memory location. Up to 19 *change-exp*s may be listed. The *change-exp* must be a pointer value if *memory-type* is a pointer; otherwise, it must be an integer value.

Change Port—Change Contents of I/O Ports

```
port-type address {[TO end-address]} = change-exp [,...]19<cr>
                  {[LENGTH n]       }
```

where

port-type is one of the following:

- PORT—references the 8-bit port value at *address*.
- WPORT—references the 16-bit port value at *address* and *address* + 1, one byte at a time and not as a single 16-bit port value.

address is the address of an 8086 port and is an integer value between 0 through 65,535 inclusive. If a range of ports is specified, *address* is the starting address of the range.

TO *end-address* specifies the upper limit of a range of port addresses. The *end-address* is an integer value between 0 and 65,535 inclusive, and must be greater than or equal to *address*. Both addresses must have a base of zero.

LENGTH n specifies the number of port or word port addresses to be displayed. The value of n must be an integer.

change-exp is the value to replace the contents of the specified port.

Display Commands

Display Register—Display Contents of 8086 Registers

```
{register[, ...]19}
{   REGISTER      } <cr>
{    FLAG         }
```

where

register is any of the following keyword references (up to 19 can be entered, separated by spaces):

8086 Register Type	Keyword References
8-bit registers	RAL, RAH, RBL, RBH, RCL, RCH, RDL, RDH
16-bit registers	RAX, RBX, RCX, RDX, SP, BP, SI, DI, SS, CS, DS, ES, IP, RF
1-bit status flags	CFL, PFL, AFL, ZFL, SFL, TFL, IFL, DFL, OFL

REGISTER is a command keyword that causes the display of all the 16-bit 8086 registers.

FLAG is a command keyword that displays all the 1-bit status flags.

Display Memory—Display 8086 Memory

memory-type address {[TO *end-address*] / [LENGTH n]} <cr>

where

memory-type is one of the following keywords: BYTE, WORD, SINTEGER, INTERGER, POINTER.

address is a pointer value that contains a base (which need not be entered if 0) and a displacement and specifies an address of a memory location. If a range is specified, *address* is the starting address in the range. The *address* can be either:

TO *end-address* specifies the upper limit of a range of memory. The *end-address* must be greater than or equal to *address*. Both addresses must have the same base.

LENGTH *n* specifies the number of bytes, words, or pointers to be displayed. The value *n* must be an integer.

Display Memory—Display 8086 Memory in ASM Form

ASM *address* {[TO *end-address*] / [LENGTH n]} <cr>

where

address is a pointer value that contains a base and a displacement and specifies an address of a memory location. If a range is specified, *address* is the starting address in the range. The *address* can be either:

TO *end-address* specifies the upper limit of a range of memory. *end-address* must be greater than or equal to *address*. Both addresses must have the same base.

LENGTH n specifies the number of bytes, words, or pointers to be displayed. The value of n is an integer.

Display Port—Display I/O Port Contents

port-type address {[TO *end-address*] / [LENGTH n]} <cr>

where

port-type is one of the following keywords:

- PORT—references the 8-bit port value at *address*.
- WPORT—references the 16-bit port value at *address* and *address* + 1, one byte at a time and not as a single 16-bit port value.

address is the address of an 8086 port and is an integer value between 0 through 65,535 inclusive. If a range is specified, *address* is the starting address of the range.

TO *end-address* specifies the upper limit of a range of port addresses. *end-address* is an integer value between 0 and 65,535 inclusive, and must be greater than or equal to *address*.

LENGTH n specifies the number of port or word port addresses to be displayed. The value n must be an integer.

Display Boolean—Display Boolean Value

```
BOOL expression<cr>
```

where

expression is an integer expression, the result of which is evaluated to a boolean value. If the least significant bit of the result equals 1, the boolean value is TRUE; otherwise the boolean value is FALSE.

Display Stack—Display User Stack Contents

```
STACK expression<cr>
```

where

expression is an integer expression, the value of which defines the number of words on the user stack to be displayed.

EVALUATE—Display Integers in Five Bases

```
EVALUATE expression [SYMBOLICALLY]<cr>
```

where

expression is an integer expression.

SYMBOLICALLY is a keyword that displays each numeric value output by the command as as a symbol or a source statement, plus a remainder.

Symbol Manipulation Commands

Define Symbol—Enter New Symbol

```
DEFINE [..module].symbol = change-exp [OF memory-type]<cr>
```

where

module is the name of an existing program module, in which *symbol* is to be located.

symbol is a user-defined symbol to be entered into the symbol table for use during the debugging session. The *symbol* references a location in the symbol table.

change-exp is a numeric expression, the value of which is to be assigned to *symbol*. The *change-exp* represents an address of statement labels or variables, or the value of a constant.

OF *memory-type* specifies any of the following: BYTE, WORD, SINTEGER, INTEGER, or POINTER. If *memory-type* is omitted, *symbol* has no type.

Display Symbols—Display One or More Symbols

```
{          SYMBOL                          }
{[..module].symbol [.symbol] ...           } <cr>
```

where

SYMBOL causes the entire DEBUG-86 symbol table to be displayed.

module is the name of the program module in which *symbol* is located.

symbol is the name of a symbol that references a location in the symbol table.

Display Lines—Display Statement Numbers

```
{LINE                                      }
{[..module]#statement-number               } <cr>
```

where

LINE is a command keyword that displays all statement numbers and associated absolute addresses in the current domain.

module is the name of the program module in which *statement-number* is located.

statement-number is the source statement number. It is a numeric constant with a default suffix that is always decimal.

Display Modules—Display Module Names

```
MODULE<cr>
```

Change Symbols—Change Value of a Symbol

```
[..module].symbol[.symbol ...] ... = change-exp [OF memory-type]<cr>
```

where

module is the name of the program module in which *symbol* is located.

symbol is the name of an existing symbol that references a location in the symbol table.

change-exp is a numeric expression of a pointer value to be assigned to *symbol* and represents either the address of statement labels or variables, or the value of a constant.

OF *memory-type* specifies the memory type of *symbol*: BYTE, WORD, SINTEGER, INTEGER, or POINTER.

Remove Symbols Command

```
REMOVE {[..module].symbol [.symbol...]19 , ...}
       {SYMBOL                                 }   <cr>
       {MODULE ..module [,..module]...         }
```

where

module is the name of an existing program module in the symbol table. Up to 19 modules can be listed at one time.

symbol is the name of an existing symbol in the symbol table. Up to 19 symbols can be listed at a time.

SYMBOL is a command modifier that deletes the entire current DEBUG-86 symbol table.

MODULE is a command modifier that deletes all the symbols and lines of the named module from the symbol and statement number tables.

Set Domain Command

```
DOMAIN ..module<cr>
```

where

DOMAIN is a command keyword that establishes a default module for source statement number references.

module is the name of an existing program module in the statement number table and is prefixed by two periods (..).

Compound Commands

REPEAT Command

```
REPEAT<cr>
 [command<cr>                          ]
 [WHILE boolean-expression<cr>         ] ...
 [UNTIL boolean-expression<cr>         ]
END<cr>
```

COUNT Command

```
COUNT arithmetic-expression<cr>
 [command<cr>                          ]
 [WHILE boolean-expression<cr>         ]  ...
 [UNTIL boolean-expression<cr>         ]
END<cr>
```

IF Command

```
IF boolean-expression [THEN]<cr>
  [command<cr>] ...
[ORIF boolean-expression [THEN]<cr>] ...
  [command<cr>] ...
[ELSE<cr>                          ]
  [command<cr>] ...]
END<cr>
```

NOTE

When error 24 occurs, an additional message is displayed:

```
STATUS=00nn
D=x T=yyy S=zzz
```

where x represents the drive number, yyy the track address, zzz the sector address, and where nn has the following meanings:

For flexible disks:

01	Deleted record
02	Data field CRC error
03	Invalid address mark
04	Seek error
08	Address error
0A	ID field CRC error
0E	No address mark
0F	Incorrect data address mark
10	Data overrun or data underrun
20	Attempt to write on Write Protect
40	Drive has indicated a Write error
80	Drive not ready

For hard disks:

01	ID field miscompare
02	Data field CRC error
04	Seek error
08	Bad sector address
0A	ID field CRC error
0B	Protocol violations
0C	Bad track address
0E	No ID address mark or sector not found
0F	Bad data field address mark
10	Format error
20	Attempt to write on write-protected drive
40	Drive has indicated a write error
80	Drive not ready

RUN Program Error Messages (8086 Execution Mode)

101. HARDWARE NOT RESPONDING (fatal error)
102. INVALID SYNTAX
103. COMMAND LINE TOO LONG
104. INSUFFICIENT MEMORY TO LOAD
105. MISMATCHED SOFTWARE/FIRMWARE
106. ILLEGAL LOAD ADDRESS
107. ILLEGAL LOAD ADDRESS
108. INVALID OBJECT FILE
117. UNRESOLVED SYMBOLS (warning)
118. RAM FAILURE (warning)
119. ROM CHECKSUM ERROR (warning)

DEBUG-86 Error Messages (8086 Execution Mode)

120. Syntax error
121. Invalid token
122. No such line

123. Inappropriate number
124. Partition bounds error
125. Symbol already exists
126. Symbol does not exist
127. Memory failure
133. Null string error
134. Memory overflow
135. Stack overflow
136. Command too complex
137. Module does not exist
139. Excessive data
141. Unsuitable execute file
142. Line too long
143. Too many partitions
147. Pointer value required
148. Integer value required
149. Differing bases

Console Command Interface Errors (8080/8085 Execution Mode)

201. Unrecognized switch
202. Unrecognized delimiter
203. Invalid syntax
206. Illegal disk label
208. Checksum error
209. Relo file sequence error
210. Insufficient memory
211. Record too long
212. Illegal relo type
213. Fixup bounds error
214. Illegal SUBMIT parameter
215. Argument too long
216. Too many parameters
217. Object record too short
218. Illegal record format
219. Phase error
220. No EOF record in object module file
221. Segment overflow during LINK operation
222. Unrecognized record in object module file
223. Fixup record pointer is incorrect
224. Illegal record sequence in object module file in LINK
225. Illegal module name specified
226. Module name exceeds 31 characters
227. Command syntax requires left parenthesis
228. Command syntax requires right parenthesis
229. Unrecognized control specified in command
230. Duplicate symbol found
231. File already exists
232. Unrecognized command
233. Command syntax requires a TO clause
234. Filename illegally duplicated in command
235. File specified in command is not a library file
236. More than 249 common segments in input files
237. Specified common segment not found in object file
238. Illegal stack content record in object file
239. No module header in input object file
240. Program exceeds 64k bytes

INDEX